

Programming  
Guide

# Keysight M9393A PXIe Performance Vector Signal Analyzer



# Notices

## Copyright Notices

© Keysight Technologies 2014 - 2016

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

## Manual Part Number

M9393-90007

## Published By

Keysight Technologies Ground Floor and Second Floor, CP-11 Sector-8, IMT Manesar – 122051 Gurgaon, Haryana, India

## Edition

Edition 2.1, May, 2016

## Regulatory Compliance

This product has been designed and tested in accordance with accepted industry standards, and has been supplied in a safe condition. To review the Declaration of Conformity, go to <http://www.keysight.com/go/conformity>

## Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE

FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR OF ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT SHALL CONTROL.

KEYSIGHT TECHNOLOGIES DOES NOT WARRANT THIRD-PARTY SYSTEMLEVEL (COMBINATION OF CHASSIS, CONTROLLERS, MODULES, ETC.) PERFORMANCE, SAFETY, OR REGULATORY COMPLIANCE, UNLESS SPECIFICALLY STATED.

## 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.

## U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at <http://www.keysight.com/find/sweula>. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish

to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

## Safety Notices

### CAUTION

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

### WARNING

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

The following safety precautions should be observed before using this product and any associated instrumentation. This product is intended for use by qualified personnel who recognize

shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product.

### 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 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.

### WARNING

Operator is responsible to maintain safe operating conditions. To ensure safe operating conditions, modules should not be operated beyond the full temperature range specified in the Environmental and physical specification. Exceeding safe operating conditions can result in shorter lifespans, improper module

performance and user safety issues. When the modules are in use and operation within the specified full temperature range is not maintained, module surface temperatures may exceed safe handling conditions which can cause discomfort or burns if touched. In the event of a module exceeding the full temperature range, always allow the module to cool before touching or removing modules from chassis.

Keysight products are designed for use with electrical signals that are rated Measurement Category I and Measurement Category II, as described in the International Electro-technical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the user documentation.

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. If the circuit is capable of operating at or above 1000V,

no conductive part of the circuit may be exposed.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance-limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

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 close 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.

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.

When fuses are used in a product, replace with the same type and rating for continued protection against fire hazard.

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.

Before performing any maintenance, disconnect the line cord and all test cables.

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. For continued protection against fire hazard, replace fuse with same type and rating.

### PRODUCT MARKINGS:



The CE mark is a registered trademark of the European Community.



Australian Communication and Media Authority mark to indicate regulatory compliance as a registered supplier.

**ICES/NMB-001  
ISM GRP.1 CLASS A**

This symbol indicates product compliance with the Canadian Interference-Causing Equipment Standard (ICES-001). It also identifies the product is an Industrial Scientific and Medical Group 1 Class A product (CISPR 11, Clause 4).



South Korean Class A EMC Declaration. This equipment is Class A suitable for professional use and is for use in electromagnetic environments outside of the home.

A 급 기기 ( 업무용 방송통신기자재 )  
이 기기는 업무용 ( A 급 ) 전자파적합기  
기로서 판 매자 또는 사용자는 이 점을 주  
의하시기 바라 며 , 가정외의 지역에서  
사용하는 것을 목적으 로 합니다.



This product complies with the WEEE Directive marketing requirement. The affixed product label (above) indicates that you must not discard this electrical/electronic product in domestic household waste.

Product Category: With reference to the equipment types in the WEEE directive Annex 1, this product is classified as "Monitoring and Control instrumentation" product. Do not dispose in domestic household waste. To return unwanted products, contact your local Keysight office, or for more information see <http://about.keysight.com/en/companyinfo/environment/takeback.shtml>



This symbol indicates the instrument is sensitive to electrostatic discharge (ESD). ESD can damage the highly sensitive components in your instrument. ESD damage is most likely to occur as the module is being installed or when cables are connected or disconnected. Protect the circuits from ESD damage by wearing a grounding strap that provides a high resistance path to ground. Alternatively, ground yourself to discharge any buildup static charge by touching the outer shell of any grounded instrument chassis before touching the port connectors.



This symbol on an instrument means caution, risk of danger. You should refer to the operating instructions located in the user documentation in all cases where the symbol is marked on the instrument.



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.

# Contents

<b>What You Will Learn in this Programming Guide</b> .....	<b>8</b>
Related Websites .....	9
Related Documentation .....	9
Additional Related Documentation .....	11
Documentation Map .....	13
Overall Process Flow .....	13
<b>Installing Hardware, Software, and Licenses</b> .....	<b>15</b>
<b>APIs for the M9393A PXIe VSA</b> .....	<b>18</b>
IVI Compliant or IVI Class Compliant .....	18
IVI Driver Types .....	19
IVI Driver Hierarchy .....	20
Instrument-Specific Hierarchies for the M9393A .....	21
Naming Conventions Used to Program IVI Drivers .....	23
<b>Creating a Project with IVI-COM Using C-Sharp</b> .....	<b>25</b>
Step 1 - Create a Console Application .....	25
Step 2 - Add References .....	25
Step 3 - Add Using Statements .....	26
To Access the IVI Drivers Without Specifying or Typing The Full Path ..	27
Step 4 - Create Instances of the IVI-COM Drivers .....	27
Step 5 - Initialize the Driver Instances .....	27
Step 5 - Initialize the Driver Instances .....	27
Initialize() Options .....	28
Initialize() Parameters .....	29
M9300A Reference Sharing .....	30
Resource Names .....	32
Step 6 - Write the Program Steps .....	33
Step 7 - Close the Driver .....	34
Step 8 - Building and Running a Complete Program Using Visual C-Sharp ...	34
Example Program 1- Code Structure .....	35
Example Program 1- How to Print Driver Properties, Check for Errors, and	
Close Driver Sessions .....	36
Disclaimer .....	39
<b>Working with PA FEM Measurements</b> .....	<b>40</b>
Test Challenges Faced by Power Amplifier Testing .....	40
Performing a Channel Power Measurement, Using Immediate Trigger .....	41
Example Program 2- Code Structure .....	42
Example Program 2 - Pseudo-code .....	43
Example Program 2 - Channel Power Measurement Using Immediate	
Trigger .....	45
Disclaimer .....	48
Performing a WCDMA Power Servo and ACPR Measurement .....	48
Example Program 3 - Code Structure .....	48

Example Program 3 - Pseudo-code .....	49
Example Program 3 - WCDMA Power Servo and ACPR Measurement ..	52
Making Harmonic Measurements with the M9393A VSA .....	57
Using the M9393A with the Resource Manager (M9000) and Modular X-Apps (M90XA) .....	58
M9393A List Mode .....	60
List Mode Set Up .....	60
M9393A Programming Advanced Topics - Hints only .....	64
Extending the RF Frequency Range (3.6 GHz to 50 GHz) .....	64
Attaining Higher IF Bandwidths Using a Wideband Digitizer .....	64
Performing Data Streaming With an External Wideband Digitizer .....	65
Understanding API for Peer-to-Peer Support .....	65
Differences between the M9391 and M9393 .....	66
Appendix - Verify Instruments Connect Pass Self-Test are Updated .....	67
Verify that VSG 1 is Connected, Passes Self-Test, and Contains up to Date Firmware .....	67
Verify that VSA 1 is Connected, Passes Self-Test, and Contains up to Date Firmware .....	67
Appendix - Using Visual Studio 2010 .....	68
Glossary .....	69
References .....	71

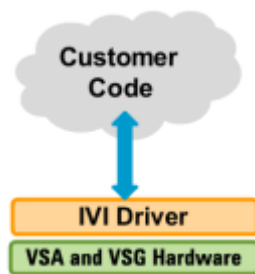


## What You Will Learn in this Programming Guide

This programming guide is intended for individuals who write and run programs to control test-and-measurement instruments. Specifically, 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
- access these Acquisition Modes: IQ, Spectrum, Stepped, Power and FFT
- control the Keysight M9393A PXIe Performance Vector Signal Analyzer (VSA) and Keysight M9381A PXIe Vector Signal Generator (VSG) while performing PA/FEM Power Measurement Production Tests
- generate waveforms created by Signal Studio software (licenses are required)



IVI-COM Console Applications that are covered in this programming guide are used to perform PA/FEM acquisition measurements with the M9393A PXIe VSA from signals that are created with the M9381A PXIe VSG.

- Example Program 1: How to Print Driver Properties, Check for Errors, and Close Driver Sessions
- Example Program 2: How to Perform a Channel Power Measurement, Using Immediate Trigger
- Example Program 3: How to Perform a WCDMA Power Servo and ACPR Measurement
- Example Program 4: How to Perform Transmitter Tests with 89600 VSA Software,(Playing Waveforms on M9381A PXIe VSGs, Using External Trigger)



## Related Websites

- [Keysight Technologies PXI and AXIe Modular Products](#)
  - [M9393A PXIe Vector Signal Analyzer](#)
  - [M9381A PXIe Vector Signal Generator](#)
- [Keysight Technologies](#)
  - [IVI Drivers & Components Downloads](#)
  - [Keysight I/O Libraries Suite](#)
  - [GPIB, USB, & Instrument Control Products](#)
  - [Keysight VEE Pro](#)
  - [Technical Support, Manuals, & Downloads](#)
  - [Contact Keysight Test & Measurement](#)
- [IVI Foundation](#) - Usage Guides, Specifications, Shared Components Downloads
- [MSDN Online](#)

## Related Documentation

To access documentation related to the Keysight M9393A PXIe Performance Vector Signal Analyzer, use one of the following methods:

- If the product software is installed on your PC, the related documents are also available in the software installation directory.

Document	Description	Default Location on 64-bit Windows system	Format
Startup Guide	Includes procedures to help you to unpack, inspect, install (software and hardware), perform instrument connections, verify operability, and troubleshoot your product.  Also includes an annotated block diagram.	C:\Program Files (x86)\Keysight\M9393\Help\M9393_StartupGuide.pdf	PDF

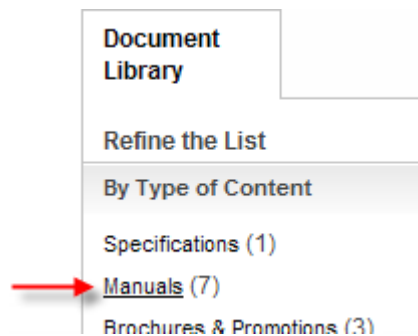
<b>Document</b>	<b>Description</b>	<b>Default Location on 64-bit Windows system</b>	<b>Format</b>
Programming Guide	Shows you how to use Visual Studio 2010 with the .NET Framework to write IVI-COM Console Applications in Visual C#.	C:\Program Files (x86)\Keysight\M9393\Help\M9393_ProgrammingGuide.pdf	PDF
IVI Driver reference (help system)	Provides detailed documentation of the IVI-COM and IVI-C driver API functions, as well as information to help you get started with using the IVI drivers in your application development environment.	C:\Program Files (x86)\Keysight\M9393\Help\AgM9393.chm	CHM (Microsoft Help Format)
Soft Front Panel (SFP) help system	Provides product introduction, tour of the SFP user interface, how-to procedures (for example, configuration, self test, operational check), and troubleshooting.	C:\Program Files (x86)\Keysight\M9393\Help\M9393_SFP_Help.chm	CHM (Microsoft Help Format)
Data Sheet	In addition to a detailed product introduction, the data sheet supplies full product specifications.	C:\Program Files (x86)\Keysight\M9393\Help\M9393_DataSheet.pdf	PDF
LabVIEW Driver Reference	Provides detailed documentation of the LabVIEW G Driver API functions.	C:\Program Files (x86)\Keysight\M9393\Help\KtM9393_LabVIEW_Help.chm	CHM (Microsoft Help Format)

Document	Description	Default Location on 64-bit Windows system	Format
Software Release Notes	Includes recent changes, enhancements, and bug fixes in the current release.	C:\Program Files (x86)\Keysight\M9393\Help\M9393_SoftwareReleaseNotes.pdf	PDF

**NOTE** Alternatively, you can find these documents under: **Start > All Programs > Keysight > M9393.**

To understand the available user documentation in context to your workflow, [click here](#).

To find the very latest versions of the user documentation, go to the product web site ([www.keysight.com/find/M9393A](http://www.keysight.com/find/M9393A)) and download the files from the Manuals support page (go to Document Library > Manuals):



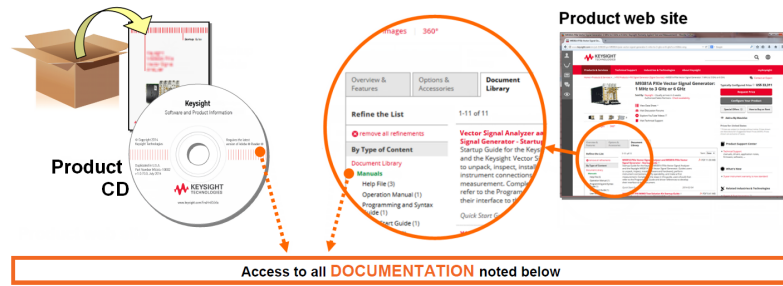
**NOTE** Depending upon your order, you may find the following product documentation useful.

### Additional Related Documentation

Document	Description	Location
M9018A PXIe Chassis Startup Guide	Provides installation and startup information for the M9018A PXIe Chassis.	<a href="http://www.keysight.com/find/M9018A">www.keysight.com/find/M9018A</a>
M9036A PXIe Embedded Controller Startup Guide (optional)	Provides installation and startup information for the M9036A PXIe Embedded Controller.	<a href="http://www.keysight.com/find/M9036A">www.keysight.com/find/M9036A</a>

Document	Description	Location
M9037A PXIe Embedded Controller Startup Guide (optional)	Provides unpacking, installation and beginning usage information for the M9037A PXIe Embedded Controller.	<a href="http://www.keysight.com/find/M9037A">www.keysight.com/find/M9037A</a>
M9021A PCIe Cable Interface Module Installation Guide (optional)	Provides installation details for the M9021A Gen 2, x8 PCIe cable interface module.	<a href="http://www.keysight.com/find/M9021A">www.keysight.com/find/M9021A</a>
Multiple PXIe and AXIe Chassis System Configuration Tool	This interactive tool helps you to configure and install PXIe and AXIe multiple chassis systems.	<a href="http://www.keysight.com/find/pxie-multichassis">www.keysight.com/find/pxie-multichassis</a>
M9381A PXIe Vector Signal Generator	Provides Startup, LabVIEW Driver, and Data sheet information of M9381 PXIe Vector Signal Generator.	<a href="http://www.keysight.com/find/M9381">www.keysight.com/find/M9381</a>
M9169E PXI Switchable-Input Programmable Step Attenuator Module	Provides Technical Overview document and Service and Service Guide.	<a href="http://www.keysight.com/find/M9169E">http://www.keysight.com/find/M9169E</a>

# Documentation Map



## Startup Guide

- Unpack product
- Verify shipment
- Install software
- Verify operation
- Troubleshooting

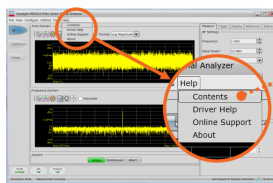
## Data Sheet

- Product description
- Technical specifications

## Programming Guide

- Product intro
- Programming Procedures
- Sample Programs

## Soft Front Panel (SFP) user interface

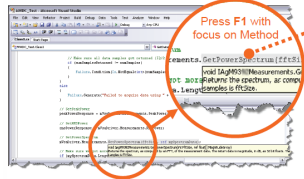


## SFP help system



- Theory of operation
- Block diagram
- Configuration
- Self test
- Operational check
- Field calibration
- Troubleshooting

## Visual Studio

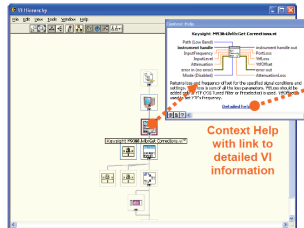


## IVI Driver help system

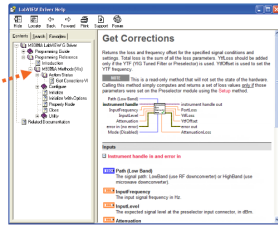


- IVI-COM and IVI-C driver reference
- Sample programs

## LabVIEW



## LabVIEW Driver help system



- LabVIEW driver reference
- Sample programs

# Overall Process Flow

To write and run programs, perform the following steps:

1. Write source code using Microsoft Visual Studio 2010 or later with .NET Visual C# running on Windows 7.
2. Compile source code using the .NET Framework Library.

3. Produce an Assembly.exe file – this file can run directly from Microsoft Windows without the need for any other programs.
  - When using the Visual Studio Integrated Development Environment (IDE), the Console Applications you write are stored in conceptual containers called Solutions and Projects.
  - You can view and access Solutions and Projects using the Solution Explorer window (View > Solution Explorer).

## Installing Hardware, Software, and Licenses

1. Unpack and inspect all hardware.
2. Verify the shipment contents.

3. Install the software. Note the following order when installing software.
  - a. Install Microsoft Visual Studio 2010 with .NET Visual C# running on Windows 7.

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

The following steps, defined in the *Keysight M9393A PXIe VSA Startup Guide*, but repeated here, must be completed before programmatically controlling the M9393A PXIe VSA hardware with IVI drivers.
  - b. Install Keysight IO Libraries Suite (IOLS), Version 16.3.17218.1 or newer; this installation includes Keysight Connections Expert.
  - c. (Required for MIMO) Install Keysight 89600 Vector Signal Analyzer Software, Version 16.2 or newer.
  - d. Install the M9393A PXIe Performance VSA driver software, Version 1.0.0.0 or newer.
  - e. Install the M938xA PXIe VSG driver software, Version 1.3.105.0 or newer.
  - f. Install the M9018A PXIe Chassis driver software, Version 1.3.443.1 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> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Keysight IO Libraries Suite Recommended.
    - <http://www.keysight.com/find/89600> (Required for MIMO) > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver that corresponds to "89600 VSA software".
    - <http://www.keysight.com/find/m9393a> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver.
    - <http://www.keysight.com/find/m9381a> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver.
    - <http://www.keysight.com/find/m9018a> > Select Technical Support > Select the Drivers, Firmware & Software tab > Download the Instrument Driver.
    - <http://www.keysight.com/find/ivi> - download other installers for Keysight IVI-COM drivers.
4. Install the hardware modules and make cable connections.
5. Verify operation of the modules (or the system that the modules create).



**NOTE**

Before programming or making measurements, conduct a Self-Test on each M9393A PXIe VSA and each M9381A PXIe VSG to make sure there are no problems with the modules, cabling, or backplane trigger mapping.

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

## APIs for the M9393A PXIe VSA

The following IVI driver terminology may be used when describing the Application Programming Interfaces (APIs) for the M9393A PXIe VSA.

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](http://www.ivifoundation.org)

IVI Instrument Classes (Defined by the IVI Foundation)

Currently, there are 13 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

The M9393A PXIe VSA does not belong to any of these 13 IVI Instrument Classes and are therefore described as "NoClass" modules.

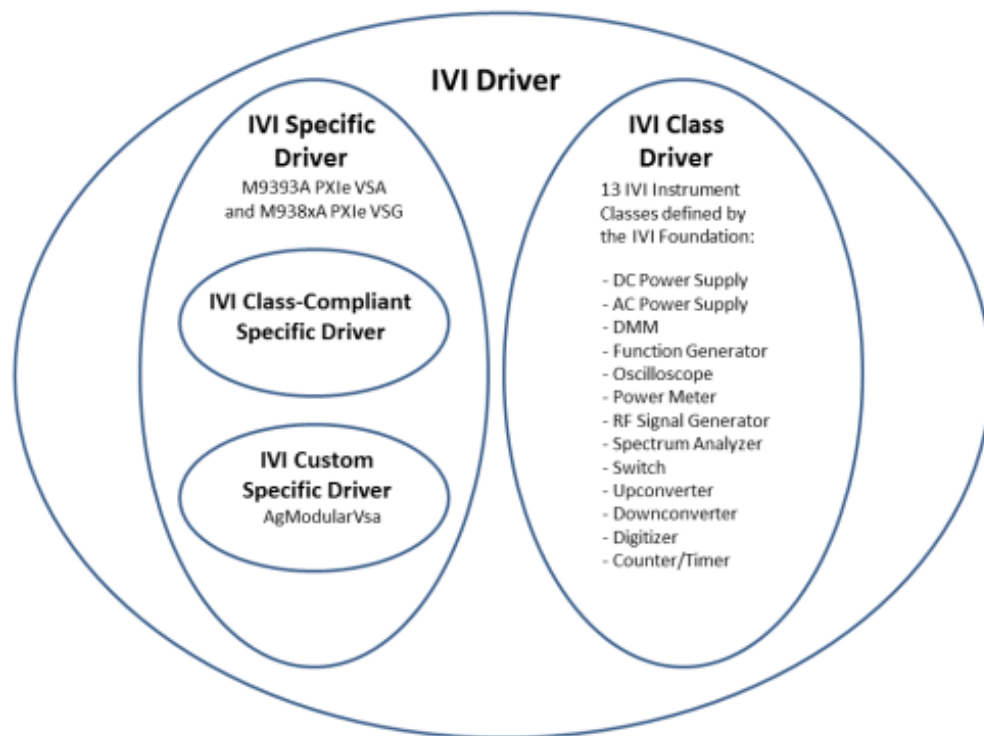
### IVI Compliant or IVI Class Compliant

The M9393A PXIe VSA is 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 Class Driver
  - Is an IVI Driver needed only for interchangeability in IVI-C environments
  - The IVI Class may be IVI-defined or customer-defined

- IVI Specific Driver
  - Is an IVI Driver that is written for a particular instrument such as the M9393A PXIe VSA
- IVI Class-Compliant Specific Driver
  - IVI Specific Driver that complies with one (or more) of the IVI defined class specifications
  - Used when hardware independence is desired
- IVI Custom Specific Driver
  - Is an IVI Specific Driver that is not compliant with any one of the 13 IVI defined class specifications
  - Not interchangeable

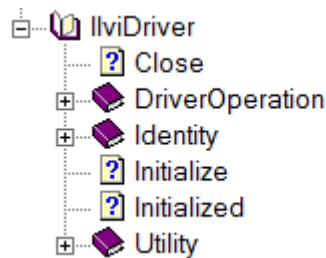
## IVI Driver Hierarchy

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

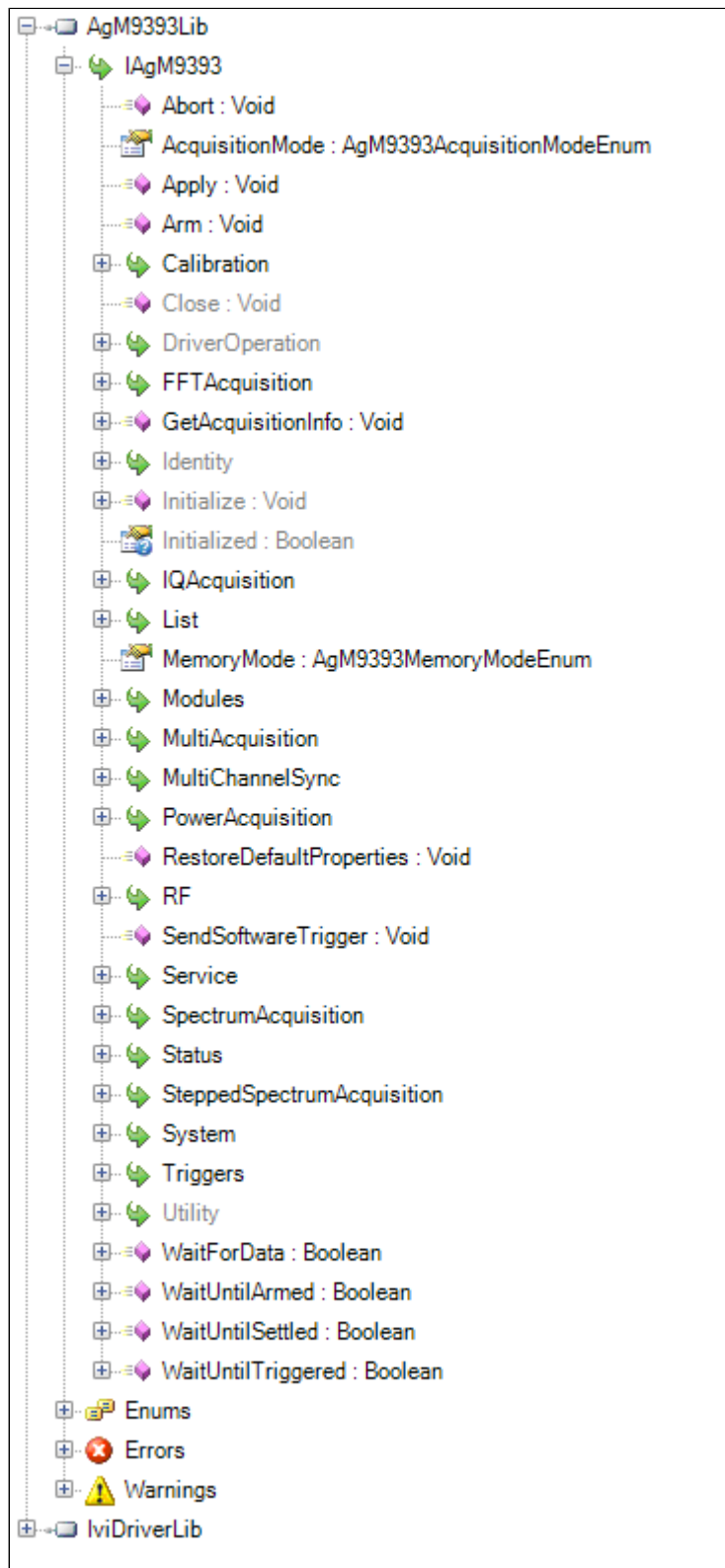
- The core of every IVI-COM driver is a single object with many interfaces.

- These interfaces are organized into two hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy – and both include the IlviDriver interfaces.
  - Class-Compliant Hierarchy - Since the M9393A PXIe VSA does not belong to one of the 13 IVI Classes, there is no Class-Compliant Hierarchy in its IVI Driver.
  - Instrument-Specific Hierarchy
    - The M9393A PXIe VSA's instrument-specific hierarchy has IAgM9393 at the root (where AgM9393 is the driver name).
      - IAgM9393 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 IlviDriver interfaces are incorporated into both hierarchies: Class-Compliant Hierarchy and Instrument-Specific Hierarchy.

The IlviDriver is the root interface for IVI Inherent Capabilities which are what the IVI Foundation has established as a set of functions and attributes that all IVI drivers must include – irrespective of which IVI instrument class the driver supports. These common functions and attributes are called IVI inherent capabilities and they are documented in IVI-3.2 – Inherent Capabilities Specification. Drivers that do not support any IVI instrument class such as the M9393A PXIe VSA must still include these IVI inherent capabilities.



## Instrument-Specific Hierarchies for the M9393A



The following table lists the instrument-specific hierarchy interfaces for M9393A PXIe VSA:

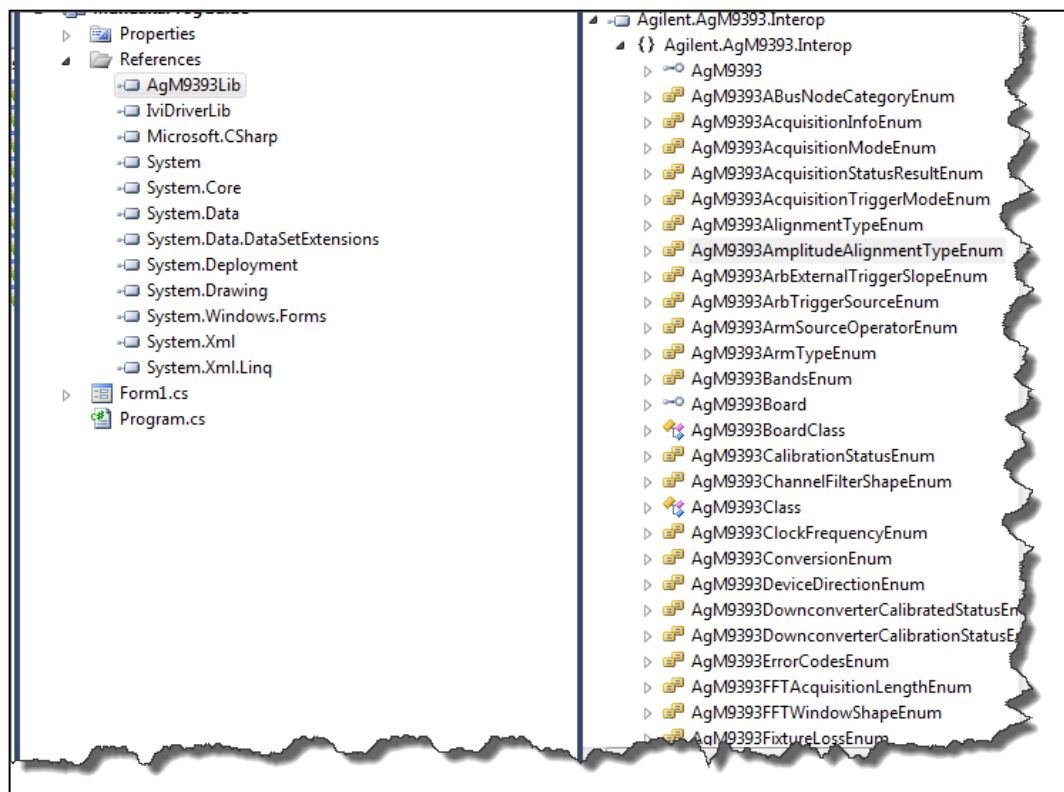
## Keysight M9393A PXIe VSA Instrument-Specific Hierarchy

AgM9393 is the driver name

IAgM9393Ex is the root interface

Perform the following steps to view the interfaces available in M9393A PXIe VSA while using Visual Studio:

1. Right-click the AgM9393Lib library file in the References folder from the Solution Explorer window.
2. 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 don't end in "Enum" but use the last word to differentiate
  - Examples: IviScopeTriggerCouplingAC and IviScopeTriggerCouplingDC



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

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 Signal or Perform a Measurement)
- 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.
2. Enter "VsaVsgProperties" as the Name of the project and click OK.  
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.
3. 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 M9393A PXIe VSA and M9381A PXIe VSG driver interfaces, references to their drivers (DLL) must be created.

1. In Solution Explorer, right-click on References and select Add Reference.
2. From the Add Reference dialog, select the COM tab.
3. Click on any of the type libraries under the "Component Name" heading and enter the letter "I".(All IVI drivers begin with IVI so this will move down the list of type libraries that begin with "I".)

**NOTE**

If you have not installed the IVI driver for the M9393A PXIe VSA and M9381A PXIe VSG products (as listed in the previous section titled "Before Programming, Install Hardware, Software, and Software Licenses"), their IVI drivers will not appear in this list.

Also, the TypeLib Version that appears will depend on the version of the IVI driver that is installed. The version numbers change over time and typically increase as new drivers are released. If the TypeLib Version that is displayed on your system is higher than the ones shown in this example, your system simply has newer versions – newer versions may have additional commands available. To get the IVI drivers to appear in this list, you must close this Add Reference dialog, install the IVI drivers, and come back to this section and repeat "Step 2 – Add References".

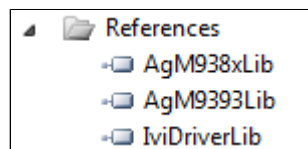
4. Scroll to the IVI section and, using Shift-Ctrl, select the following type libraries then select OK.

IVI AgM938x 1.2 Type Library  
 IVI AgM9393 1.0 Type Library

**NOTE**

When any of the references for the AgM9393A or AgM938x 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 Specifying or Typing The Full Path

Add the following using statements to your program so you don't have to specify the entire path when using the drivers:

```
using Ivi.Driver.Interop;  
using Keysight.AgM938x.Interop;  
using Keysight.AgM9393.Interop;
```

#### 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 M9393A PXIe VSA and M9381A PXIe VSG are both considered NoClass modules (because they do not belong to one of the 13 IVI Classes), the COM Session Factory is not used to create instances of their IVI-COM drivers. So, the M9393A PXIe VSA and M938xA PXIe VSG IVI-COM drivers use direct instantiation. Because direct instantiation is used, their IVI-COM drivers may not be interchangeable with other VSA and VSG modules.

#### To Create Driver Instances

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

```
IAGM9393 VsaDriver = new AgM9393();  
IAGM9381 VsgDriver = new AgM9381();
```

#### Step 5 - Initialize the Driver Instances

##### 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.

## Initialize() Options

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

- `string ResourceName = PXI[bus]::device[::function][::INSTR]`  
*Example:* `string ResourceName = "PXI13::0::0::INSTR; PXI14::0::0::INSTR; PXI15::0::0::INSTR; PXI16::0::0::INSTR";`  
*Description:* `VsgResourceName` or `VsaResourceName` – The driver is typically initialized using a physical resource name descriptor, often a VISA resource descriptor.
- `bool IdQuery = true;`  
*Description:* `IdQuery` – Setting the `IdQuery` to false prevents the driver from verifying that the connected instrument is the one the driver was written for because if `IdQuery` is set to true, this will query the instrument model and fail initialization if the model is not supported by the driver.
- `bool Reset = true;`  
*Description:* `Reset` – Setting `Reset` to true tells the driver to initially reset the instrument.
- `string OptionString = "QueryInstrStatus=true, Simulate=true,`  
*Description:* `OptionString` – Setup the following initialization options:
  - `QueryInstrStatus=true` (Specifies whether the IVI specific driver queries the instrument status at the end of each user operation.)
  - `Simulate=true` (Setting `simulate` to true tells the driver that it should not attempt to connect to a physical instrument, but use a simulation of the instrument instead.)
  - `Cache=false` (Specifies whether or not to cache the value of properties.)
  - `InterchangeCheck=false` (Specifies whether the IVI specific driver performs interchangeability checking.)
  - `RangeCheck=false` (Specifies whether the IVI specific driver validates attribute values and function parameters.)
  - `RecordCoercions=false` (Specifies whether the IVI specific driver keeps a list of the value coercions it makes for `ViInt32` and `ViReal64` attributes.)
- `DriverSetup= Trace=false";`  
*Description:* `DriverSetup=` (This is used to specify settings that are supported by the driver, but not defined by IVI. If the Options String parameter (`OptionString` in this example) contains an assignment for the Driver Setup attribute, the `Initialize` function assumes that everything following '`DriverSetup=`' is part of the assignment.)

Model=VSG or Model=VSA (Instrument model to use during simulation.)  
Trace=false (If false, an output trace log of all driver calls is not saved in an XML file.)

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

AgM938x IVI Driver Reference

Start > All Programs > Keysight IVI Drivers > IVI-COM-C Drivers > AgM938x Source > AgM938x IVI Driver Help

AgM9393 IVI Driver Reference

Start > All Programs > Keysight IVI Drivers > IVI-COM-C Drivers > AgM9393A VSA > AgM9393A IVI Driver Help

Initialize() Parameters

**NOTE**

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

```
// The M9300A PXIe Reference should be included as one
of the modules in
// either the M9381A PXIe VSG configuration of modules
or the M9393A PXIe VSA configuration of modules.
//
// If the M9300A PXIe Reference is only included in one
configuration,
// that configuration should be initialized first.
// See "Understanding M9300A Frequency Reference
Sharing" .

string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::
0::INSTR;PXI12::0::0::INSTR;PXI13::0::0::INSTR";
string VsaResourceName = "PXI14::0::0::INSTR;PXI10::0::
0::INSTR;PXI9::0::0::INSTR;

bool IdQuery = true;
bool Reset = true;

string VsgOptionString = "QueryInstrStatus=true,
Simulate=false, DriverSetup= Model=VSG, Trace=false";
string VsaOptionString = "QueryInstrStatus=true,
Simulate=false, DriverSetup= Model=VSA, Trace=false";

// Initialize the drivers
VsgDriver.Initialize(VsgResourceName, IdQuery, Reset,
VsgOptionString);
Console.WriteLine("VSG Driver Initialized");
```

```
VsaDriver.Initialize(VsaResourceName, IdQuery, Reset,
VsaOptionString);
Console.WriteLine("VSA Driver Initialized");
```

```
#region Initialize Driver Instances
string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::0::INSTR;PXI12::0::0::INSTR;PXI13::0::0::INSTR";
string VsaResourceName = "PXI14::0::0::INSTR;PXI10::0::0::INSTR;PXI9::0::0::INSTR";

bool IdQuery = true;
bool Reset = true;

string VsgOptionString = "QueryInstrStatus=true, Simulate=false, DriverSetup= Model=VSG, Trace=false";
string VsaOptionString = "QueryInstrStatus=true, Simulate=false, DriverSetup= Model=VSA, Trace=false";

VsgDriver.Initialize(VsgResourceName, IdQuery, Reset, VsgOptionString);
Console.WriteLine("VSG Driver Initialized");

VsaDriver.Initialize(VsaResourceName, IdQuery, Reset, VsaOptionString);
Console.WriteLine("VSA Driver Initialized");
void IAgModularVsa.Initialize(string ResourceName, bool IdQuery, bool Reset, string OptionString)
#endregion
```

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.

### M9300A Reference Sharing

The M9300A PXIe Reference can be shared by up to five configurations of modules that can be made up of the M9393A PXIe VSA or the M9381A PXIe VSG or both. The M9300A PXIe Reference must be included as one of the modules in at least one of these configurations. The configuration of modules that is initialized first must include the M9300A PXIe Reference so that the other configurations that depend on the reference signal get the signal they are expecting. If the configuration of modules that is initialized first does not include the M9300A PXIe Reference, unlock errors will occur.

Example: M9300A PXIe Reference with M9381A PXIe VSG

The M9381A PXIe VSG should be initialized first before initializing the VSA if:

- M9381A PXIe VSG configuration of modules includes:
  - M9311A PXIe Modulator
  - M9310A PXIe Source Output
  - M9301A PXIe Synthesizer
  - M9300A PXIe Reference

Note that the M9300A PXIe Reference is part of the M9381A PXIe VSG configuration of modules.

```
string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::0::INSTR;
PXI12::0::0::INSTR;PXI13::0::0::INSTR"
```

- M9393A PXIe VSA configuration of modules includes:
  - M9308A PXIe Synthesizer
  - M9365A PXIe Downconverter
  - M9214A PXIe IF Digitizer

```
string VsaResourceName = "PXI14::0::0::INSTR;PXI10::0::0::INSTR;
PXI9::0::0::INSTR";
```

Example: M9300A PXIe Reference with M9393A PXIe VSA

The M9393A PXIe VSA should be initialized first before initializing the M9381A PXIe VSG if:

- M9381A PXIe VSG configuration of modules includes:
  - M9311A PXIe Modulator
  - M9310A PXIe Source Output
  - M9301A PXIe Synthesizer

```
string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::0::
INSTR;PXI12::0::0::INSTR";
```

- M9393A PXIe VSA configuration of modules includes:
  - M9300A PXIe Reference\*  
Note that the M9300A PXIe Reference is part of the M9393A PXIe VSA configuration of modules.
  - M9308A PXIe Synthesizer
  - M9365A PXIe Downconverter
  - M9214A PXIe IF Digitizer

```
string VsaResourceName = "PXI14::0::0::INSTR;PXI10::0::0::
INSTR;PXI9::0::0::INSTR;PXI13::0::0::INSTR";
```

Example: M9300A PXIe Reference Shared With Both Modules

The M9393A PXIe VSA or the M9381A PXIe VSG can be initialized first since the M9300A PXIe Reference is included in both configurations of modules:

- M9381A PXIe VSG configuration of modules includes:
  - M9311A PXIe Modulator
  - M9310A PXIe Source Output
  - M9301A PXIe Synthesizer
  - M9300A PXIe Reference\*  
Note that the M9300A PXIe Reference is part of the M9381A PXIe VSG configuration of modules.

```
string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::0::INSTR;
PXI12::0::0::INSTR";PXI13::0::0::INSTR;
```

- M9393A PXIe VSA configuration of modules includes:
  - M9300A PXIe Reference\*  
Note that the M9300A PXIe Reference is part of the M9393A PXIe VSA configuration of modules.
  - M9308A PXIe Synthesizer
  - M9365A PXIe Downconverter
  - M9214A PXIe IF Digitizer

```
string VsaResourceName = "PXI14::0::0::INSTR;PXI10::0::0::INSTR;  
PXI9::0::0::INSTR;PXI13::0::0::INSTR;
```

### Resource Names

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

```
string VsaResourceName = "%";  
  
string VsgResourceName = "%";
```

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, Keysight Command Expert, National Instruments Measurement and Automation Explorer (MAX), or you can use the Keysight product's Soft Front Panel (SFP) to get the physical Resource Name string.

For example, you might get the following Resource Name address string.

```
string VsgResourceName = "PXI8::0::0::INSTR;PXI11::0::0::  
INSTR;PXI12::0::0::INSTR;PXI13::0::0::INSTR";
```

Where the module arrangement in slots is as shown below:

ModuleName	M9311A PXIe Modulator	M9310A PXIe Source Output	M9301A PXIe Synthesizer	M9300A PXIe Reference
Slot Number	2	4	5	6
VISA Address	PXI8::0::0::INSTR;	PXI11::0::0::INSTR;	PXI12::0::0::INSTR;	PXI13::0::0::INSTR;

(The slot used by a particular module can be determined from the Connect to Instrument dialog.)  
Similarly, you might use the M9393A Soft Front Panel to construct the Resource Name address string.



## Step 6 - Write the Program Steps

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

### Using the Soft Front Panel to Write Program Commands

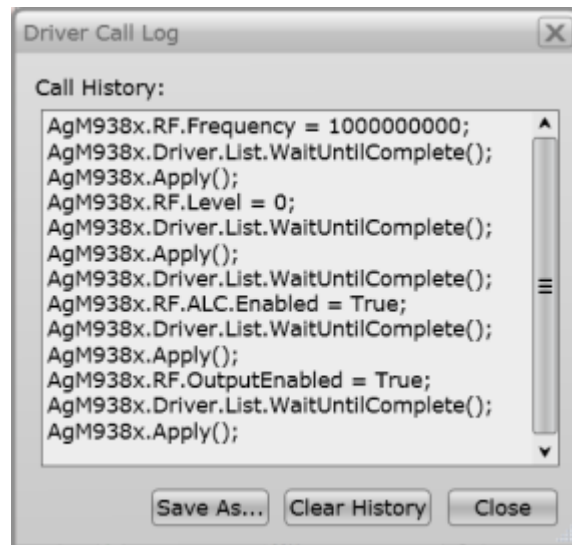
You may find it useful when developing a program to use the instrument's Soft Front Panel (SFP) "Driver Call Log"; this driver call log is used to view a list of driver calls that have been performed when changes are made to the controls on the soft front panel.

In this example, open the Soft Front Panel for the M938xA PXIe VSG and perform the following steps:

1. Set the output frequency to 1 GHz.
2. Set the output level to 0 dBm.
3. Enable the ALC.
4. Enable the RF Output.

AgM938x is the driver name used by the SFP. VsgDriver 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".

```
IAgM938x VsgDriver = new AgM938x(); |
```



```
// Set the output frequency to 1 GHz
VsgDriver.RF.Frequency = 1000000000;
// Set the output level to 0 dBm
VsgDriver.RF.Level = 0;
// Enables the ALC
VsgDriver.ALC.Enabled = true;
// Enables the RF Output
VsgDriver.RF.OutputEnabled = true;
```

```

// Waits until the list is finished or the specified
time passes
bool retval = VsgDriver.List.WaitUntilComplete();

//&hellip;or you could use the following:

// Waits 100 ms until output is settled before
producing signal
bool retval = VsgDriver.RF.WaitUntilSettled(100);

```

## 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(VsaDriver!= null && VsaDriver.Initialized)
    {
        // Close the VSA driver
        VsaDriver.Close();
        Console.WriteLine("VSA Driver Closed\n");
    }

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

```

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

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

1. Open the solution file `SolutionNameThatYouUsed.sln` in Visual Studio 2010.
2. Set the appropriate platform target for your project.
  - In many cases, the default platform target (Any CPU) is appropriate.
  - However, if you are using a 64-bit PC (such as Windows 7) to build a .NET application that uses a 32-bit IVI-COM driver, you may need to specify your project's platform target as x86.

3. Choose Project > ProjectNameThatYouUsed Properties and select Build | Rebuild Solution.

- Tip: You can also do the same thing from the Debug menu by selecting Start Debugging or pressing the F5 key.

Example programs are available from C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgM9393\Examples

**NOTE**

Before you run the LiveSoftFrontPanel C# example program, ensure that the associated app.config file has the following content for the .NET Runtime interoperability:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,
Version=v4.0" />
    <supportedRuntime version="v2.0.50727/>
  </startup>
</configuration>
```

### Example Program 1- Code Structure

The following example code builds on the previously presented "*Tutorial: Creating a Project with IVI-COM Using C#*" and demonstrates how to instantiate two 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.

Example programs may be found by selecting: C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgM9393\Examples

```

 3  Specify using Directives
12
13  namespace VsaVsgProperties
14  {
15      class Program
16      {
17          static void Main(string[] args)
18          {
19              // Create driver instances
20              IAgM938x M9381driver = new AgM938x();
21              IAgM9393 M9393driver = new AgM9393();
22          try
23          {
24              Initialize Driver Instances
41
42              Print Driver Properties
61
62              Perform Tasks
66
67              Check for Errors
86
87          }
88          catch (Exception ex)
89          {
90              Console.WriteLine(ex.Message);
91          }
92          finally
93          {
94              Close Driver Instances
107
108
109          Console.WriteLine("Done - Press Enter to Exit");
110          Console.ReadLine();
111      }
112  }
113 }
114
115 /*

```

Example Program 1- How to Print Driver Properties, Check for Errors, and Close Driver Sessions

```

// Copy the following example code and compile it as a
// C# Console Application
// Example__VsaVsgProperties.cs
#region Specify using Directives
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Ivi.Driver.Interop;
using Agilent.AgM938x.Interop;
using Agilent.AgM9393.Interop;
#endregion

namespace VsaVsgProperties

```

```

{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instances
            IAgM938x VsgDriver = new AgM938x();
            IAgM9393 VsaDriver = new AgM9393();

            try
            {
                #region Initialize Driver Instances
                string VsgResourceName = "PXI8::0::0::INSTR;
PXI11::0::0::INSTR;PXI12::0::0::INSTR;PXI13::0::0::
INSTR";
                string VsaResourceName = "PXI14::0::0::INSTR;
PXI10::0::0::INSTR;PXI9::0::0::INSTR";

                bool IdQuery = true;
                bool Reset    = true;

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

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

                VsgDriver.Initialize(VsgResourceName, IdQuery,
Reset, VsgOptionString);
                Console.WriteLine("VSG Driver Initialized");

                VsaDriver.Initialize(VsaResourceName, IdQuery,
Reset, VsaOptionString);
                Console.WriteLine("VSA Driver Initialized\n\n");
                #endregion

                #region Print Driver Properties
                // Print IviDriverIdentity properties for the
                PXIe VSG
                Console.WriteLine("Identifier:  {0}", VsgDriver.
Identity.Identifier);
                Console.WriteLine("Revision:    {0}", VsgDriver.
Identity.Revision);
                Console.WriteLine("Vendor:      {0}", VsgDriver.
Identity.Vendor);
                Console.WriteLine("Description: {0}", VsgDriver.
Identity.Description);
                Console.WriteLine("Model:      {0}", VsgDriver.
Identity.InstrumentModel);
                Console.WriteLine("FirmwareRev: {0}", VsgDriver.
Identity.InstrumentFirmwareRevision);
                Console.WriteLine("Simulate:   {0}\n",
VsgDriver.DriverOperation.Simulate);
            }
        }
    }
}

```

```

        // Print IviDriverIdentity properties for the
        PXIe VSA
        Console.WriteLine("Identifier:  {0}", VsaDriver.
Identity.Identifier);
        Console.WriteLine("Revision:    {0}", VsaDriver.
Identity.Revision);
        Console.WriteLine("Vendor:     {0}", VsaDriver.
Identity.Vendor);
        Console.WriteLine("Description: {0}", VsaDriver.
Identity.Description);
        Console.WriteLine("Model:      {0}", VsaDriver.
Identity.InstrumentModel);
        Console.WriteLine("FirmwareRev: {0}", VsaDriver.
Identity.InstrumentFirmwareRevision);
        Console.WriteLine("Simulate:   {0}\n",
VsaDriver.DriverOperation.Simulate);
        #endregion

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

        #region Check for Errors
        // Check VSG instrument for errors
        int VsgErrorNum = -1;
        string VsgErrorMsg = null;
        while (VsgErrorNum != 0)
        {
            VsgDriver.Utility.ErrorQuery(ref
VsgErrorNum, ref VsgErrorMsg);
            Console.WriteLine("VSG ErrorQuery: {0}, {1}
\n", VsgErrorNum, VsgErrorMsg);
        }

        // Check VSA instrument for errors
        int VsaErrorNum = -1;
        string VsaErrorMsg = null;
        while (VsaErrorNum != 0)
        {
            VsaDriver.Utility.ErrorQuery(ref
VsaErrorNum, ref VsaErrorMsg);
            Console.WriteLine("VSA ErrorQuery: {0}, {1}
\n", VsaErrorNum, VsaErrorMsg);
        }
        #endregion
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

```

finally
{
    if (VsgDriver != null && VsgDriver.Initialized)
    {
        // Close the driver
        VsgDriver.Close();
        Console.WriteLine("VSG Driver Closed");
    }
    if (VsaDriver != null && VsaDriver.Initialized)
    {
        // Close the driver
        VsaDriver.Close();
        Console.WriteLine("VSA Driver Closed\n");
    }
}

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

```

## Disclaimer

© 2015 Keysight Technologies Inc. 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.

## Working with PA FEM Measurements

The RF front end of a product includes all of the components between an antenna and the baseband device. The purpose of an RF front end is to upconvert a baseband signal to RF that can be used for transmission by an antenna. An RF front end can also be used to downconvert an RF signal that can be processed with ADC circuitry. As an example, the RF signal that is received by a cellular phone is the input into the front end circuitry and the output is a down-converted analog signal in the intermediate frequency (IF) range. This down-converted signal is the input to a baseband device, an ADC. For the transmit side, a DAC generates the signal to be up-converted, amplified, and sent to the antenna for transmission. Depending on whether the system is a Wi-Fi, GPS, or cellular radio will require different characteristics of the front end devices.

RF front end devices fall into a few major categories: RF Power Amplifiers, RF Filters and Switches, and FEMs [Front End Modules].

- RF Power Amplifiers and RF Filters and Switches typically require the following:
  - PA [Power Amplifier] – Production Tests which include:
    - Channel Power – Power Acquisition Mode is used to return one value back through the API.
    - ACPR [Adjacent Channel Power Ratio] – When making fast ACPR measurements, "Baseband Tuning" is used to digitally tune the center frequency in order to make channel power measurements, at multiple offsets, using the Power Acquisition interface.
    - Servo Loop- When measuring a power amplifier, one of the key measurements is performing a Servo Loop because when you measure a power amplifier:
      - it is typically specified at a specific output power
      - there is a need to adjust the source input level until you measure the exact power level - to do this, you will continually adjust the source until you achieve the specified output power then you make all of the ACPR and harmonic parametric measurements at that level.
  - FEMs [Front End Modules] – which could be a combination of multiple front end functions in a single module or even a "Switch Matrix" that switches various radios (such as Wi-Fi, GSM, PCS, Bluetooth, etc.) to the antenna.

## Test Challenges Faced by Power Amplifier Testing

The following are the test challenges faced by power amplifier testing:

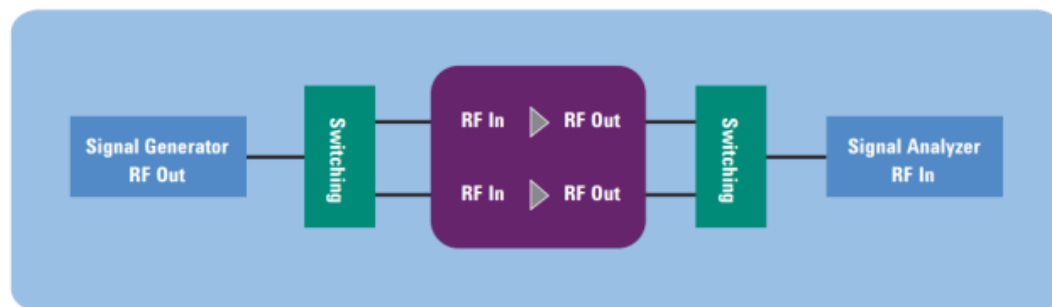
- The need to quickly adjust power level inputs to the device under test (DUT).
- The need to assess modulation performance (i.e., ACPR and EVM) at high output power levels.



The figure below shows a simplified block diagram for the M9381A PXIe VSG and M9393A PXIe VSA in a typical PA / FEM test system.

Typical power amplifier modules require an input power level of 0 to + 5 dBm, digitally modulated according to communication standards such as WCDMA or LTE. The specified performance of the power amplifier or front end module is normally set at a specific output level of the DUT. If the devices have small variations in gain, it may be necessary to adjust the power level from the M9381A PXIe VSG to get the correct output level of the DUT. Only after the DUT output level is set at the correct value can the specified parameters be tested. The time spent adjusting the M9381A PXIe VSG to get the correct DUT output power can be a major contributor to the test time and the overall cost of test.

The M9381A PXIe VSG is connected to the DUT using a cable and switches. The switching may be used to support testing of multi-band modules or multi-site testing. The complexity of the switching depends on the number of bands in the devices and the number of test sites supported by the system. The DUTs are typically inserted into the test fixture using an automated part handler. In some cases, several feet of cable is required between the M9381A PXIe VSG and the input of the DUT.



The combination of the RF cables and the switching network can add several dB of loss between the output of the M9381A PXIe VSG and the input of the DUT, which requires higher output levels from the M9381A PXIe VSG. Since the tests are performed with a modulated signal, the M9381A PXIe VSG must also have adequate modulation performance at the higher power levels.

### Performing a Channel Power Measurement, Using Immediate Trigger

Standard	Sample Rate	Channel Filter Type	Channel Filter Parameter	Channel Filter Bandwidth	Channel Offsets
WCDMA	5 MHz	RRC	0.22	3.84 MHz	5, 10 MHz
LTE 10 MHz FDD	11.25 MHz	Rectangular	N/A	9 MHz	10, 20 MHz

Standard	Sample Rate	Channel Filter Type	Channel Filter Parameter	Channel Filter Bandwidth	Channel Offsets
LTE 10 MHz TDD	11.25 MHz	Rectangular	N/A	9 MHz	10, 20 MHz
1xEV-DO	2 MHz	RRC	0.22	1.23 MHz	1.25, 2.5 MHz
TD-SCDMA	2 MHz	RRC	0.22	1.28 MHz	1.6, 3.2 MHz
GSM/EDGE Channel	1.25 MHz	Gaussian	0.3	271 KHz	
GSM /EDGEORFS	1.25 MHz	TBD	TBD	30 KHz	400, 600 KHz

### Example Program 2- Code Structure

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

1. Send RF and Power Acquisition commands to the M9393A PXIe VSA driver and Apply changes to hardware,
2. Check the instrument queue for errors.
3. Perform a Channel Power Measurement,
4. Report errors if any occur, and close the drivers.

Example programs may be found by selecting: *C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgM9393\Examples*

```

// Copy the following example code and compile it as a C# Console Application
// Example_ChannelPowerImmTrigger_Alternate.cs
// Channel Power Measurement, Using Immediate Trigger
Specify using Directives

namespace ChannelPowerImmTrigger
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instance
            IAgM9393 M9393driver = new AgM9393();

            try
            {
                Initialize Driver Instance

                Check Instrument Queue for Errors

                Receiver Settings

                Run Commands

            }
            catch (Exception ex)
            {
                Console.WriteLine("Exceptions for the drivers:\n");
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Close Driver Instances

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

```

Example Program 2 - Pseudo-code

Example Program 2 - Pseudo-code

Initialize Driver for VSA, Check for Errors

- Send RF Settings to VSA Driver:
  - Frequency
  - Level
  - Peak to Average Ratio
  - Conversion Mode
  - IF Bandwidth
  - Set Acquisition Mode to "Power"
- Send Power Acquisition Setting to VSA Driver:

- Sample Rate
- Duration
- Channel Filter
- Apply Method to Send Changes to Hardware
  - Wait for Hardware to Settle
- Send Arm Method to VSA
- Send Read Power Method to VSA

Close Driver for VSA

## Using FFT Acquisition Mode in Channel Power Measurements

### Advantages of using FFT Acquisition Mode in Channel Power Measurements

The FFT Acquisition mode uses a feature in the digitizer DSP to do a high speed averaged FFT on the fly. The resultant FFT data is processed in fast capture memory in hardware. Although this FFT process is limited in length, it is very good for producing very fast channel power measurements. This process differs with the Spectrum Acquisition mode, which does an FFT in the host controller.

Baseband tuning within the digitizer is another feature of using FFT Acquisition mode for making channel power measurements. Baseband tuning can be made digitally inside the digitizer to shift off the center frequency. In addition, while doing this, the local oscillator in the downconverter doesn't have to move, while you are doing baseband tuning. Tuning digitally inside the digitizer has the advantage that it does not require any settling time. This offset frequency matches the offset frequency on all the other acquisition modes.

### Pseudo code

The following pseudo code is a modification of the Example Program 2 - Pseudo-code. Use this and Example Program 2 - Channel Power Measurement Using Immediate Trigger as a reference for coding.

#### Initialize Driver for VSA, Check for Errors

- Send RF Settings to VSA Driver
  - Frequency: (range 9 kHz to 27 GHz)
  - Level: (range -170 dBm to 24 dBm)
  - Peak to Average Ratio: (range 0 dBm to 20 dBm)
  - Conversion Mode: (choose between Single High Side, Single Low Side, Image Protect, or Auto)
  - IF Bandwidth: (choose between 40 MHz and 160 MHz)
  - Set Acquisition Mode to "FFT"
- Send FFT Acquisition Setting to VSA Driver
  - FFT Length: select 64, 128, 256, or 512. Typically use 256
  - Sample Rate:
    - Usable bandwidth of FFT is 80% of sample rate

- Bandwidth should be large enough to include all channels to be measured
- Window Shape: typically use Hann for best results
- Offset Frequency: (range -160 MHz to 160 MHz)
- FFT Averaging Count: (range 1 to 65536)
- Apply Method to Send Changes to Hardware
  - Wait for Hardware to Settle
- Send Arm Method to VSA
- Send Read Power Method to VSA

Close Driver for VSA

## Example Program 2 - Channel Power Measurement Using Immediate Trigger

```
// Copy the following example code and compile it as a
// C# Console Application
// Example_ChannelPowerImmediateTrigger.cs
#region Specify using Directives
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using Ivi.Driver.Interop;
    using Agilent.AgM9393.Interop;
#endregion

namespace ChannelPowerImmTrigger
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instances
            VsaDriver = new AgM9393();

            try
            {
                #region Initialize Driver Instances
                    string VSAResourceName = "PXI14::0::0::
INSTR;PXI10::0::0::INSTR;PXI9::0::0::INSTR;PXI13::0::0::
INSTR";

                    bool IdQuery = true;
                    bool Reset = true;

                    string VSAOptionString = "QueryInstrStatus=t
rue, Simulate=false, DriverSetup= Model=M9393A,
Trace=false";
```

```

        VsaDriver.Initialize(VSAResourceName,
IdQuery, Reset, VSAOptionString);
        Console.WriteLine("VSA Driver Initialized\n"
);
    #endregion

    #region Check Instrument Queue for
Errors
        // Check VSA instrument for errors
        int VsaErrorNum = -1;
        string VsaErrorMsg = null;
        while (VsaErrorNum != 0)
        {
            VsaDriver.Utility.ErrorQuery(ref
VsaErrorNum, ref VsaErrorMsg);
            Console.WriteLine("VSA ErrorQuery: {0},
{1}\n", VsaErrorNum, VsaErrorMsg);
        }
    #endregion

    #region Receiver Settings
        // Receiver Settings
        double Frequency = 2000000000.0;
        double Level = 5;
        double RmsValue = 5;
        double ChannelTime = 0.0001;
        double MeasureBW = 5000000.0;
        AgM9393ChannelFilterShapeEnum FilterType =
AgM9393ChannelFilterShapeEnum.
AgM9393ChannelFilterShapeRootRaisedCosine;
        double FilterAlpha = 0.22;
        double FilterBw = 3840000.0;
        double MeasuredPower = 0;
        bool Overload = true;
    #endregion

    #region Run Commands
        // Setup the RF Path in the Receiver
        VsaDriver.RF.Frequency = Frequency;
        VsaDriver.RF.Power = Level;
        VsaDriver.RF.Conversion =
AgM9393ConversionEnum.AgM9393ConversionAuto;
        VsaDriver.RF.PeakToAverage = RmsValue;
        VsaDriver.RF.IFBandwidth = 40000000.0; //
Use IF filter wide enough for all adjacent channels
        // Configure the Acquisition
        VsaDriver.AcquisitionMode =
AgM9393AcquisitionModeEnum.AgM9393AcquisitionModePower;
        VsaDriver.PowerAcquisition.Bandwidth =
MeasureBW; // 5 MHz
        VsaDriver.PowerAcquisition.Duration =
ChannelTime; // 100 us

```

```

        VsaDriver.PowerAcquisition.ChannelFilter.
Configure(FilterType, FilterAlpha, FilterBw);
        // Send Changes to hardware
        VsaDriver.Apply();
        VsaDriver.WaitUntilSettled(100);

        string response = "y";
        while (string.Compare(response, "y") == 0) {
            Console.WriteLine("Press Enter to Run
Test");

            Console.ReadLine();

            VsaDriver.Arm();
            VsaDriver.PowerAcquisition.ReadPower(0,
AgM9393PowerUnitsEnum.AgM9393PowerUnitsdBm, ref
MeasuredPower, ref Overload);
            Console.WriteLine("Measured Power: " +
MeasuredPower + " dBm");
            Console.WriteLine(String.Format("Overloa
d = {0}", Overload ? "true" : "false"));
            Console.WriteLine("Repeat? y/n");
            response = Console.ReadLine();
        }
        #endregion

    }
    catch (Exception ex)
    {
        Console.WriteLine("Exceptions for the drivers:
\n");
        Console.WriteLine(ex.Message);
    }
    finally
    #region Close Driver Instances
    {
        if (VsaDriver != null && VsaDriver.Initialized)
        {
            // Close the driver
            VsaDriver.Close();
            Console.WriteLine("VSA Driver Closed\n");
        }
    }
    #endregion

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

```

## Disclaimer

© 2015 Keysight Technologies Inc. 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.

## Performing a WCDMA Power Servo and ACPR Measurement

When making a WCDMA Power Servo and ACPR measurement, Servo is performed using "Baseband Tuning" to adjust the source amplitude and then "Baseband Tuning" is used to digitally tune the center frequency in order to make channel power measurements, at multiple offsets, using the Power Acquisition interface of the M9393A PXIe VSA.

### NOTE

The M9393A PXIe VSA and the M9381A PXIe VSG offers two modes for adjusting frequency and amplitude:

- **RF Tuning** – allows the M9381A PXIe VSG to be set across the complete operating frequency and amplitude range
- **Baseband Tuning** – allows the frequency and amplitude to be adjusted within the IF bandwidth (160 MHz) and over a range of the output level.

## Example Program 3 - Code Structure

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

1. Send RF and Modulation commands to the M9381A PXIe VSG driver and Apply changes to hardware.
2. Send RF and Power Acquisition commands to the M9393A PXIe VSA driver and Apply changes to hardware.
3. Run a Servo Loop until it is at the required output power from DUT.
4. Perform an ACPR Measurement for each Adjacent Channel to be measured.



5. Check drivers for errors and report the errors if any occur, and close the drivers.

Example programs may be found by selecting: *C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgM9393\Examples*

```
// Copy the following example code and compile it as a C# Console Application
// Example_PaServoAcpr_Alternate.cs
// WCDMA Power Servo and ACPR Measurement
Specify using Directives

namespace PaServoAcpr
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instances
            IAgM938x M9381driver = new AgM938x();
            IAgM9393 M9393driver = new AgM9393();

            try
            {
                Initialize Driver Instances

                Check Instrument Queue for Errors

                Create Default Settings for WCDMA Uplink Signal

                Run Commands

            }
            catch (Exception ex)
            {
                Console.WriteLine("Exceptions for the drivers:\n");
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Close Driver Instances

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

Example Program 3 - Pseudo-code

Initialize Drivers for VSG and VSA, Check for Errors

- Send RF Settings to VSG Driver:
  - Frequency
  - RF Level to Maximum Needed
  - RF Enable On
  - ALC Enable Off (for baseband power changes)

- Send Modulation Commands to VSG Driver:
  - Load WCDMA Signal Studio File
  - Enable Modulation
  - Play ARB File
  - Set ARB Scale to 0.5
  - Set Baseband Power Offset to -10 dB
- Apply Method to Send Changes to Hardware
  - Wait for Hardware to Settle
- Send RF Settings to VSA Driver:
  - Frequency
  - Level
  - Peak to Average Ratio
  - Conversion Mode
  - IF Bandwidth
  - Set Acquisition Mode to "Power"
- Send Power Acquisition Setting to VSA Driver:
  - Sample Rate
  - Duration
  - Channel Filter
- Apply Method to Send Changes to Hardware
  - Wait for Hardware to Settle

#### Servo Loop:

- Set Baseband Power Offset on VSG to expected value
- Send Apply Method to VSG
- Send Arm Method to VSA
- Send ReadPower Method to VSA
- Repeat Until at Required Output Power from DUT
- Last Reading is Channel Power Measurement

#### ACPR Measurement:

- Set Acquisition Duration Property on VSA to Value for Adjacent Channel Measurements
- Set Frequency Offset Property on VSA to Channel Offset Frequency
- Send Apply Method to VSA
- Send Arm Method to VSA
- Send ReadPower Method to VSA

- Repeat for each Adjacent Channel to be Measured

### Example Program 3 - WCDMA Power Servo and ACPR Measurement

```
// Copy the following example code and compile it as a
C# Console Application
// Example__PaServoAcpr.cs
// WCDMA Power Servo and ACPR Measurement
#region Specify using Directives
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using Ivi.Driver.Interop;
    using Agilent.AgM938x.Interop;
    using Agilent.AgM9393.Interop;
#endregion

namespace PaServoAcpr
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create driver instances
            IAgM938x VsgDriver = new AgM938x();
            IAgM9393 VsaDriver = new AgM9393();

            try
            {
                #region Initialize Driver Instances
                string VsgResourceName = "PXI8::0::0::INSTR;
PXI11::0::0::INSTR;PXI12::0::0::INSTR;PXI13::0::0::
INSTR";
                string VsaResourceName = "PXI14::0::0::
INSTR;PXI10::0::0::INSTR;PXI9::0::0::INSTR";

                bool IdQuery = true;
                bool Reset = true;

                string VsgOptionString = "QueryInstrStatus=t
rue, Simulate=false, DriverSetup= Model=VSG,
Trace=false";

                string VsaOptionString = "QueryInstrStatus=t
rue, Simulate=false, DriverSetup= Model=VSA,
Trace=false";

                VsaDriver.Initialize(VsaResourceName,
IdQuery, Reset, VsaOptionString);
                Console.WriteLine("VSA Driver Initialized\n"
);
            }
        }
    }
}
```

```

VsgDriver.Initialize(VsgResourceName,
IdQuery, Reset, VsgOptionString);
    Console.WriteLine("VSG Driver Initialized");

#endregion

#region Check Instrument Queue for Errors
    // Check VSG instrument for errors
    int VsgErrorNum = -1;
    string VsgErrorMsg = null;
    while (VsgErrorNum != 0)
    {
        VsgDriver.Utility.ErrorQuery(ref
VsgErrorNum, ref VsgErrorMsg);
        Console.WriteLine("VSG ErrorQuery: {0},
{1}", VsgErrorNum, VsgErrorMsg);
    }

    // Check VSA instrument for errors
    int VsaErrorNum = -1;
    string VsaErrorMsg = null;
    while (VsaErrorNum != 0)
    {
        VsaDriver.Utility.ErrorQuery(ref
VsaErrorNum, ref VsaErrorMsg);
        Console.WriteLine("VSA ErrorQuery: {0},
{1}\n", VsaErrorNum, VsaErrorMsg);
    }
#endregion

#region Create Default Settings for WCDMA
Uplink Signal
    // Source Settings
    double Frequency = 1000000000.0;
    double Level = 3;
    // If a Signal Studio waveform file is
used, it may require a software license.
    string ExamplesFolder = "C:Program Files
(x86)AgilentM938xExample Waveforms";
    string WaveformFile = "WCDMA_UL_DPCHH_2DPDCH
_1C.wfm";
    string FileName = ExamplesFolder +
WaveformFile;
    string ArbRef = "Mod Waveform";

    // Receiver Settings
    double ChannelTime = 0.0001;
    double AdjacentTime = 0.0005;
    double IfBandwidth = 40000000.0;
    double PowerOffset = 0;
    double MeasureBW = 5000000.0;

```

```

        AgM9393ChannelFilterShapeEnum FilterType =
AgM9393ChannelFilterShapeEnum.
AgM9393ChannelFilterShapeRootRaisedCosine;
        double FilterAlpha = 0.22;
        double FilterBw = 3840000.0;
        double[] FreqOffset = new double[] {-5000000
.0, 5000000.0, -10000000.0, 10000000.0};

        double MeasuredPower = 0;
        bool Overload = true;
        double MeasuredChannelPower;
        bool ChannelPowerOverload;
        double[] MeasuredACPR = new double[4];
        double SampleRate = 0;
        double RmsValue = 0;
        double ScaleFactor = 0;
#endregion

#region Run Commands
        // These commands are sent to the VSG
Driver, "Apply" or "PlayArb" methods send to hardware
        VsgDriver.RF.Frequency = Frequency;
        VsgDriver.RF.Level = Level;
        VsgDriver.RF.OutputEnabled = true;
        VsgDriver.ALC.Enabled = false;
        VsgDriver.Modulation.IQ.UploadArbAgilentFile
(ArbRef, FileName);
        VsgDriver.Modulation.Enabled = true;
        VsgDriver.Modulation.BasebandPower = -10;
        // Play the ARB, sending all changes to
hardware
        VsgDriver.Modulation.PlayArb(ArbRef,
AgM938xStartEventEnum.AgM938xStartEventImmediate);
        VsgDriver.Modulation.Scale = 0.5;
        VsgDriver.Apply();

        // Get the Sample Rate and RMS Value (Peak
to Average Ratio) of the Current Waveform
        AgM938xMarkerEnum RfBlankMarker =
AgM938xMarkerEnum.AgM938xMarkerNone;
        AgM938xMarkerEnum AlcHoldMarker =
AgM938xMarkerEnum.AgM938xMarkerNone;
        VsgDriver.Modulation.IQ.ArbInformation
(ArbRef, ref SampleRate, ref RmsValue, ref ScaleFactor,
ref RfBlankMarker, ref AlcHoldMarker);

        // Setup the RF Path in the Receiver
        VsaDriver.RF.Frequency = Frequency;
        VsaDriver.RF.Power = Level + PowerOffset;
        VsaDriver.RF.Conversion =
AgM9393ConversionEnum.AgM9393ConversionAuto;
        VsaDriver.RF.PeakToAverage = RmsValue;
        VsaDriver.RF.IFBandwidth = IfBandwidth;

```

```

        // Configure the Acquisition
        VsaDriver.AcquisitionMode =
AgM9393AcquisitionModeEnum.AgM9393AcquisitionModePower;
        VsaDriver.PowerAcquisition.Bandwidth =
MeasureBW;
        VsaDriver.PowerAcquisition.Duration =
ChannelTime;
        VsaDriver.PowerAcquisition.ChannelFilter.
Configure(FilterType, FilterAlpha, FilterBw);
        // Send Changes to hardware
        VsaDriver.Apply();
        VsaDriver.WaitUntilSettled(100);

        string response = "y";
        while (string.Compare(response, "y") == 0) {
            Console.WriteLine("Press Enter to Run
Test");

            Console.ReadLine();

            // Run a group of baseband power
commands to change the source level and make a power
measurement at each step.
            // Simulates Servo loop timing, but
does not use the measured power to adjust the next
source level
            VsaDriver.PowerAcquisition.Duration =
ChannelTime;
            VsaDriver.Apply();
            double[] LevelOffset = new double[] {-3,
-2, -1, -0.5, -0.75};
            for (int Index = 0; Index < LevelOffset.
Length - 1; Index++) {
                VsgDriver.Modulation.BasebandPower
= LevelOffset[Index];
                VsgDriver.Apply();
                VsaDriver.Arm();
                VsaDriver.PowerAcquisition.ReadPower
(0, AgM9393PowerUnitsEnum.AgM9393PowerUnitsdBm, ref
MeasuredPower, ref Overload);
            }

            // Loop Through the channel offset
frequencies for an ACPR measurement
            // Use the last value of the servo loop
for the channel power
            MeasuredChannelPower = MeasuredPower;
            ChannelPowerOverload = Overload;
            VsaDriver.PowerAcquisition.Duration =
AdjacentTime;
            for (int Index = 0; Index < FreqOffset.
Length; Index++) {
                VsaDriver.PowerAcquisition.
OffsetFrequency = FreqOffset[Index];

```

```

        VsaDriver.Apply();
        VsaDriver.Arm();
        VsaDriver.PowerAcquisition.ReadPower
(0, AgM9393PowerUnitsEnum.AgM9393PowerUnitsdBm, ref
MeasuredPower, ref Overload);
        MeasuredACPR[Index] = MeasuredPower
- MeasuredChannelPower;
    }

    // Make sure the VSA frequency offset
is back to 0 (on repeat)
    VsaDriver.PowerAcquisition.
OffsetFrequency = 0;
    VsaDriver.Apply();
    if (ChannelPowerOverload == true) {
        Console.WriteLine("Channel Power
Measurement Overload");
    }
    Console.WriteLine("Channel Power: {0}
dBm", MeasuredChannelPower);
    Console.WriteLine("ACPR1 L: {0} dBc",
MeasuredACPR[0]);
    Console.WriteLine("ACPR1 U: {0} dBc",
MeasuredACPR[1]);
    Console.WriteLine("ACPR2 L: {0} dBc",
MeasuredACPR[2]);
    Console.WriteLine("ACPR2 U: {0} dBc",
MeasuredACPR[3]);

    Console.WriteLine("Repeat? y/n");
    response = Console.ReadLine();
    }
#endregion

}
catch (Exception ex)
{
    Console.WriteLine("Exceptions for the drivers:
\n");
    Console.WriteLine(ex.Message);
}
finally
#region Close Driver Instances
{
    if (VsgDriver != null && VsgDriver.Initialized)
    {
        // Close the driver
        VsgDriver.Close();
        Console.WriteLine("VSG Driver Closed");
    }

    if (VsaDriver != null && VsaDriver.Initialized)
    {

```



```

        // Close the driver
        VsaDriver.Close();
        Console.WriteLine("VSA Driver Closed\n");
    }

}
#endregion

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

```

## Making Harmonic Measurements with the M9393A VSA

Making harmonics and spurious measurements is a key application of the M9393A Vector Signal Analyzer (VSA).

### Spectrum Acquisition mode

Harmonic measurements are made in the Spectrum Acquisition mode, which allows:

- Setting specific Span and Resolution Bandwidth (RBW)
- Averaging to "Time Peak" for peak detection
- Noise Correction
- Digital image rejection

### Considerations when making a harmonics measurement

- Span and RBW: use specified values. Modulated signals measurements will vary a large amount based on the value of the RBW.
- Widow Type: typically use HDR Flat Top.
- Averages: Set time required to achieve desired repeatability. Time for each average is FFT Size / Sample Rate.
- Overlap: Typically use 0.5.
- Results: Array of power data in units of dBm. Start Frequency and Frequency Delta between points in the array.

### Programming considerations

The following M9393A software drivers are used in constructing a program to make a harmonics measurement:

IAGM9393

Apply

Arm

Initialize

RF

- Configure
- Conversion
- Frequency
- IFBandwidth
- Power

#### AcquisitionTrigger

- Delay
- Mode
- Timeout
- TimeoutMode

#### ExternalTrigger

- Slope
- Source

#### AcquisitionMode

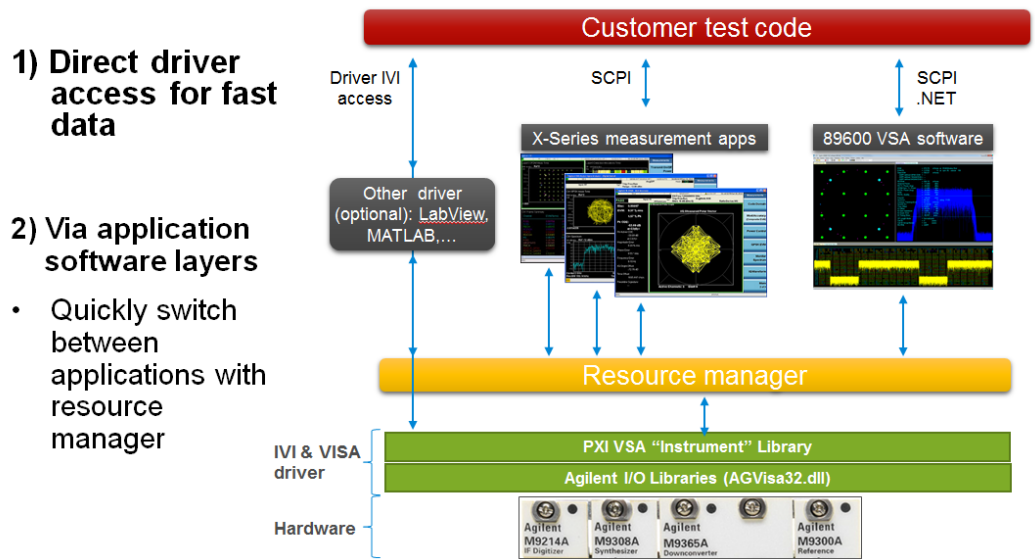
#### SpectrumAcquisition

- Averaging
  - Configure
  - Count
  - Duration
  - Mode
  - Overlap
- GetComplexSpectrum
- ResolutionBandwidth
- Span

## Using the M9393A with the Resource Manager (M9000) and Modular X-Apps (M90XA)

### Resource Manager

The Resource Manager is a program that enables common hardware to be accessed by switching between multiple programs. The image below shows common hardware (in green) being accessed directly by drivers or two programs. The Resource Manager (in yellow) in this example allows rapid switching between the 89600 VSA program and the X-Series Apps program. By being able to switch between different programs and by direct IVI driver access, test developers can gain insights and then optimize their test algorithms.



### Modular X-Series Apps

Features of the X-Series Apps are:

- Consistent interface
- Common algorithms, programming commands and shared library of measurement applications across X-Series signal analyzers and M9393A PXI VSA ensure consistent, repeatable results.
- Same look and feel as bench-top analyzer applications
- Same measurement algorithms
- Same SCPI interface
- One license supports up to four PXI VSAs

There are presently X-Series Apps for cellular communications, wireless connectivity, and general purpose apps.

The cellular communications X-Apps include the EVM measurement, which is useful for testing power amplifiers.

## M9393A List Mode

Use the "List" Mode to record a set of IVI commands that can be quickly executed on the M9393A hardware. In the List Mode, IVI commands are recorded as a binary stream and are played back using the FPGA component of the M9214A PXIe Digitizer. Thus, the execution speed for such a binary stream is much faster than if the commands were to be executed sequentially by the host controller.

List Mode is useful in automated manufacturing situations where making high speed signal acquisitions with accurate timing is important to reduce test times for a device under test (DUT). By reducing the test times, increased production throughput is achieved.

### List Mode Set Up

Using IVI methods and properties, you can create and play one or more named List that can be cataloged in the M9393A driver. A List executes a predefined set of Acquisitions, each of which starts with the specified *Acquisition Mode* and with the specific configurations, such as sample rate, offset, and frequency. There are several different *Acquisition Triggers*. Besides all the triggers available in the non-list mode, there is another acquisition trigger mode called *Scheduler*. The Scheduler makes it possible to make a series of acquisitions with accurate timing using the internal timer. The List can also generate outgoing triggers. Using proper acquisition triggers and outgoing triggers, the List can interact with DUT and other external equipment as well as control the timing of acquisitions autonomously with the internal timer. This enables you to perform complicated test sequences.

Perform the following steps:

1. Define a named List with `List.Create` and `List.End`.
2. In between `List.Create` and `List.End`, define one or more entries or Steps with `List.Entry.Add` and `List.Entry.End`.
3. In each Entry, use the `AcquisitionMode` attribute to specify what kind of acquisition is to be made, just as you do outside the List. In the List, these three modes are allowed: `IQAcquisition`, `SpectrumAcquisition`, and `PowerAcquisition`.
4. Call `List.Acquisition.Add` to define an acquisition. This method must be called only once for an entry because each entry has to define one and only one acquisition.
5. Add other methods and properties to configure an entry, as required.

#### NOTE

Properties undefined in an "Entry" will hold the same values as those from the preceding Entry, or from the start of the list definition if they are not defined in any of the preceding Entries.

6. To specify how the List advances from one entry to the other, use `Acquisition Trigger`.

Once an acquisition is finished in an entry, the List makes any changes

needed for the next acquisition in the next entry and gets to the armed state as soon as possible. You can still put `Apply` and `Arm` in the List definition, but these will be simply ignored.

7. Typically, a List contains more than one acquisition. Each acquisition is identified by a unique key or Capture ID. You need to remember or make note of the Capture ID returned by `List.Acquisition.Add`, or use the utility method `List.Acquisition.GetCaptureID(string ListName, string EntryName)`. When you need to read the data, you must pass in the correct Capture ID to the Read methods.
8. Memory must be allocated for all the acquisitions before playback. The M9393A driver performs this task automatically so that you are not required to do anything. However, in some cases, you might be required to explicitly control the timing of the memory allocation, for example, when the memory allocation process takes a long time (more than a second) for a long list. Use the `List.Acquisition.AllocateMemory` method for memory allocation in such a scenario. For details, see the M9393A IVI Documentation. Also, after the acquisition, the driver does not automatically release the memory. Use the `List.Acquisition.ReleaseMemory` method to manually release the memory, in case required. However, it is not recommended to release and re-allocate memory for each playback of a list, but rather to keep and reuse the same allocation for multiple playbacks of the same list. By default, the same memory allocation is used for subsequent playbacks, which also means that the data related to a playback gets overwritten when subsequent playback starts. Thus, you must ensure that you retrieve the required data related to acquisitions before starting the next playback.
9. Once List playback starts, acquisitions are completed one by one from the beginning of the List. You can retrieve the data from the completed acquisitions regardless of whether the entire list has been completely executed or not.  
`List.WaitForData` takes Capture ID and returns if the corresponding acquisition has been completed or not. Following this call, you can call the corresponding read methods even while the list is still playing.

See the M9393A IVI Documentation for the complete list of IVI commands.

The following sample code shows how to define a list, then to play it, and then finally to retrieve acquisition data:

```
const int SAMPLE_COUNT = 1000000;
/***** DEFINE A LIST *****/
const string LIST_NAME = "My List";

driver.List.Create(LIST_NAME);

// Do IQ acquisition in Step 1
const string STEP_1 = "Step 1";
```

```

driver.List.Entry.Add(STEP_1);
driver.List.Acquisition.Add(0);
driver.AcquisitionMode = AgM9393AcquisitionModeEnum.
AgM9393AcquisitionModeIQ;
driver.IQAcquisition.SampleRate = 5e7;
driver.IQAcquisition.SampleSize = AgM9393SampleSizeEnum.
AgM9393SampleSize64Bits;
driver.IQAcquisition.Samples = SAMPLE_COUNT;
driver.List.Entry.End();

// Do Power acquisition in Step 2
const string STEP_2 = "Step 2";
driver.List.Entry.Add(STEP_2);
driver.List.Acquisition.Add(0);
driver.AcquisitionMode = AgM9393AcquisitionModeEnum.
AgM9393AcquisitionModePower;
driver.PowerAcquisition.Duration = 1e-4;
driver.List.Entry.End();

driver.List.End();

/***** PLAY A LIST *****/
driver.List.Play(LIST_NAME, 10000,
AgM9393ListStartEventEnum.
AgM9393ListStartEventImmediate);

/***** RETRIEVE DATA *****/

// Get ID for the IQ acquisition
int idForIQ = driver.List.Acquisition.GetCaptureID
(LIST_NAME, STEP_1);

// Wait until the data becomes ready.
while (driver.List.WaitForData(idForIQ, 100) ==
AgM9393ListDataDispositionEnum.
AgM9393ListDataDispositionWaiting) {}

// Retrieve IQ data
double[] iq = new double[SAMPLE_COUNT * 2];
bool overload = false;
driver.IQAcquisition.ReadIQData(idForIQ,
                                AgM9391IQUnitsEnum.
AgM9391IQUnitsSquareRootMilliWatts,
                                0,
SAMPLE_COUNT,
                                ref iq,
                                ref overload);

// Get ID for Power acquisition

```

```

int idForPower = driver.List.Acquisition.GetCaptureID
(LIST_NAME, STEP_2);

// Wait until the data becomes ready.
while (driver.List.WaitForData(idForPower, 100) ==
AgM9393ListDataDispositionEnum.
AgM9393ListDataDispositionWaiting) { }

// Retrieve Power
double power = 0.0;
driver.PowerAcquisition.ReadPower(idForPower,
AgM9391PowerUnitsEnum.
AgM9393PowerUnitsdBm,
ref power,
ref overload);

```

**NOTE**

The M9393A IVI Documentation provides information about the IVI commands for defining Lists, performing acquisitions, playing back Lists, and scheduling acquisitions.

#### Limitations in List Mode

- Since the list is recorded with alignments valid for specific temperature and environment, temperature fluctuations can invalidate the contents of the list because calibration coefficients that are a function of temperature would be incorrect. Thus, it is recommended to re-record the list by calling `List.UpdateCorrections` if the temperature changes more than 5 degrees Celsius or if 24 hours have elapsed since the last recording. This also means that the List cannot be shared across sessions, or between hardware sets without re-recording.
- When configuring List mode in a hardware configuration involving M9169E Programmable Attenuator, the following IVI properties cannot be recorded and played back:
  - `RF.Power`
  - `RF.MixerLevelOffset`
  - `RF.InputEnabled`
An error is reported when a list entry that includes any of these properties is closed.

For more information about extended frequency range, refer to [Extending the RF Frequency Range](#).

## M9393A Programming Advanced Topics - Hints only

This section provides quick tips for performing the following functions:

- [Extending the RF Frequency Range](#)
- [Attaining Wideband IF and Using a Wideband Digitizer](#)
- [Performing Data Streaming Using an External Wideband Digitizer 1](#)
- [Understanding API for Peer-to-Peer Support](#)

### Extending the RF Frequency Range (3.6 GHz to 50 GHz)

To extend the frequency range to 50 GHz (when option M9365A-FRX is installed), it is recommended to use the M9169E Programmable Attenuator module to provide range control and automated field alignments and self-test. The RF signal is applied to the Aux 2 In connector on the M9365A Downconverter directly or through the M9169E Programmable Attenuator. For the prerequisites and setup information for these configurations, refer to the *M9393A Startup Guide*.

When writing programs for extended frequency range, use the following option string in the `Initialize()` method:

```
DriverSetup= Aux2In=true
```

This option string is used when the input signal is directly applied to the Aux 2 In connector. When using M9169E to apply the adjusted input to the Aux 2 In connector, do not specify the above option string. The resource description in the `Initialize()` method that contains information about the M9169E module automatically takes into consideration the use of Aux 2 In connector.

The *M9393A SFP Guide* explains the settings required to extend the frequency range using the M9393A Soft Front Panel.

#### NOTE

For the List Mode limitations when operating in extended frequency range, refer to [M9393A List Mode](#).

### Attaining Higher IF Bandwidths Using a Wideband Digitizer

To achieve wideband IF, the IF Conditioning/Filtering path in the M9365A Downconverter module is bypassed and an external digitizer, such as M9202A or M9703A is used to sample the signal with wider IF bandwidth. The M9393A-WB1 option must be installed to achieve this functionality. For more information on setup and cabling, refer to the *M9393A Startup Guide*.

When writing programs for the above configuration, use the following option string in the `Initialize()` method:

```
DriverSetup= ExternalDigitizer=true
```

The RF2 interface includes the following two additional properties:

- `IFFilterBypassEnabled`: Set this property to true to bypass IF filters in the IF signal path of the M9365A Downconverter module. This allows a wide IF bandwidth. By default, this property is set to false. However, when the



option `ExternalDigitizer` is set to true in the `Initialize` method this property acquires a value of true.

RF Frequency must be greater than 3.6 GHz to enable IF Filter Bypass. On a 50 GHz configuration enabled by option M9365A-FRX, RF Frequency must be greater than or equal to 3.6 GHz. Double downconversion Image Protect conversion type, which is not available above 3.6 GHz, cannot be set when IF Filter Bypass is enabled.

- `IFFilterBypassFrequency`: The center frequency in Hz of the downconverter's IF output when IF filter bypass is enabled. This property can be set when the option `ExternalDigitizer` is set to true in the `Initialize` method and the `IFFilterBypassEnabled` property is set to true. The valid range is 300 MHz to 600 MHz. The default value is 500 MHz

## Performing Data Streaming With an External Wideband Digitizer

Data streaming depends on wideband digitizer functionality being enabled in the M9393A. When using M9202A Digitizer, for instance, all the programmatic control for data streaming is done using the M9392A IVI interfaces once the M9393A is set up. M9393A IVI interfaces do not implement data streaming in any way.

## Understanding API for Peer-to-Peer Support

Peer-to-peer streaming enables M9393A to produce and transfer finite-length IQ data directly from the M9214A Digitizer to another module that supports peer-to-peer data transfer, such as the M9451A module, without involving the host controller.

### NOTE

Currently, the M9214A Digitizer does not support peer-to-peer transfer of continuous gap-free data streams.

The peer-to-peer streaming is achieved using the `PeerToPeerPort` interface of the M9393A IVI driver. Currently, M9393A supports a single port; however, in the future, additional ports could be supported. For more information about this interface, refer to the *M9393A IVI documentation*.

The peer-to-peer feature is demonstrated in the *Keysight RF PA/FEM Characterization & Test, Reference Solution* that uses PXIe M9393A VSA and PXIe M9381A VSG measurement hardware, along with the M9451A FPGA measurement accelerator module to perform DPD computations. In this Reference Solution, the peer-to-peer capability is utilized to perform the following two data transfers:

1. M9393A VSA to M9451 FPGA
2. M9451A FPGA to M9381A VSG

Using the peer-to-peer capability, the host controller is not used and hence, the data transfer becomes faster.

## Differences between the M9391 and M9393

The major differences between the M9391 and M9393 are as follows:

- M9393 supports noise correction, M9391 does not.
- M9393 supports digital image rejection, M9391 does not.
- M9391 has a dual-downconversion path across most of the band, M9393 only has this in the narrowband.
- M9393 has a Stepped Spectrum IVI interface, M9391 does not.
- M9393 has a MultiAcquisition mode in addition to the List mode, M9391 does not.
- M9393 has two IF filter bandwidths, where M9391 has only one.
- M9393 has three different offset adjustments: IF, digitizer, and mixer, whereas M9391 only has two.
- The interface for alignments is slightly different: there are more options with M9393. Additionally, the alignments are stored in EEPROM, alignments therefore don't need to be re-run in between IVI sessions.
- M9393 has a Live SFP, M9391 does not.
- M9393 requires that the RF input have no incident signal when performing alignments, M9391 does not.
- M9393 has a broadband trigger in addition to the other triggers present for the M9391.

## Appendix – Verify Instruments Connect Pass Self-Test are Updated

Before you attempt to programmatically control any hardware and make measurements, connect to each of the instrument soft front panels, one at a time, perform self-test, and verify their FPGA firmware is fully updated. If any firmware updates are made, perform the self-test again.

In the following procedures, each instrument connection must be verified, each instrument must pass self-test, and each instrument's firmware version should be checked and updated if needed.

### Verify that VSG 1 is Connected, Passes Self-Test, and Contains up to Date Firmware

1. Select Start > All Programs > Keysight > M938x > M9381 SFP and run the soft front panel of the M9381A PXIe VSG - connect to VSG #1 and the M9300A PXIe Reference.
2. Run self-test.
3. Check firmware and update if necessary.
4. Close the Firmware Update dialog box if no firmware updates are necessary. If firmware updates are required, install the updates, shut down the computer, cycle power on the M9018A PXIe Chassis, and repeat this procedure to verify connection, self-test, and no further firmware updates are necessary.

### Verify that VSA 1 is Connected, Passes Self-Test, and Contains up to Date Firmware

1. Select Start > All Programs > Keysight > M9393 > M9393 SFP and run the soft front panel of the M9393A PXIe VSA - connect to VSA #1 and the M9300A.
2. Run self-test.
3. Check firmware and update if necessary.
4. Close the Firmware Update dialog box if no firmware updates are necessary. If required, install the updates, shut down the computer, cycle power on the M9018A PXIe Chassis, and repeat this procedure to verify connection, self-test, and no further firmware updates are necessary.

## Appendix - Using Visual Studio 2010

Microsoft Visual Studio 2010 has slight differences compared to Visual Studio 2008 in creating projects.

COM Interop Libraries that are added as References to projects in Visual Studio 2010 default to *Embed Interop Types* : *True* which leads to compilation errors. This usually leads to two errors on the following line of code, C# is used as an example.

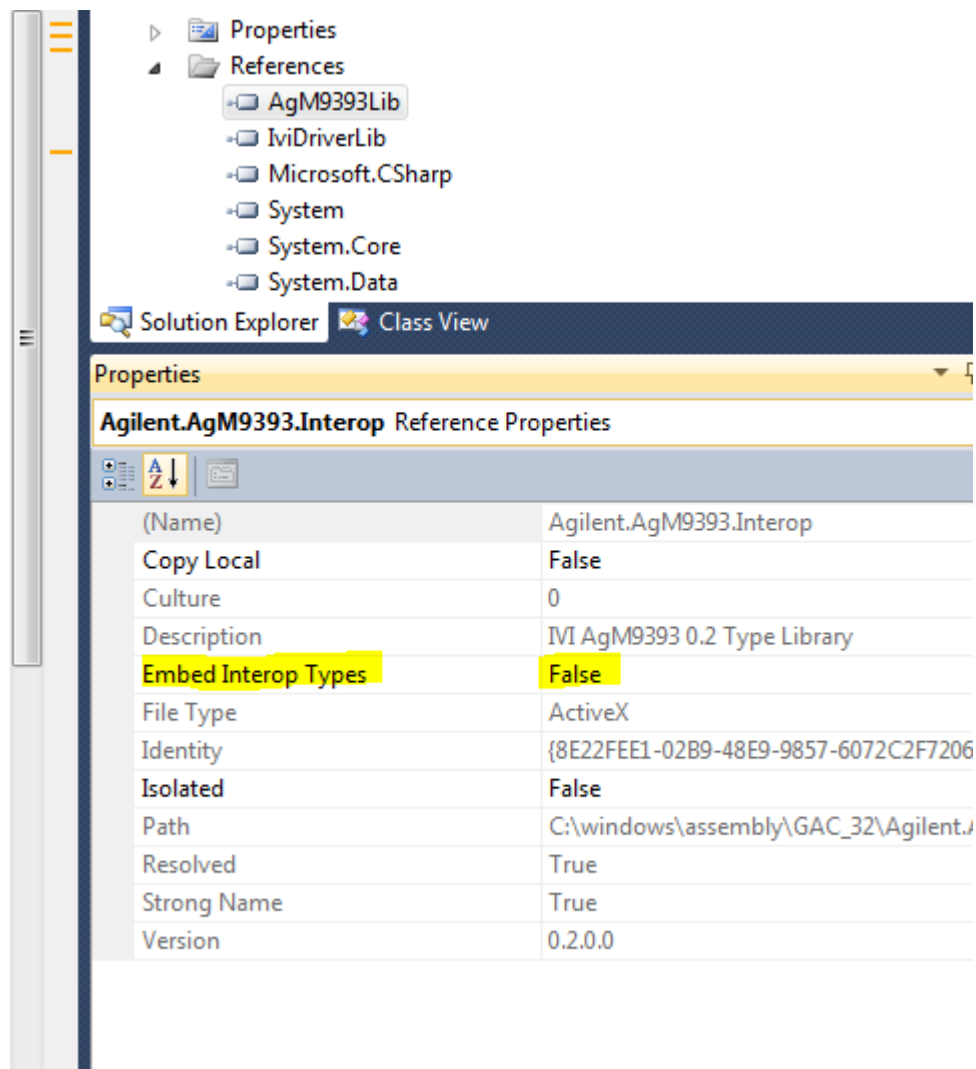
```
AgM9393 driver = new AgM9393Class();
```

The compilation errors would be:

The type 'Agilent.AgM9393.Interop.AgM9393Class' has no constructors defined.

Interop type 'Agilent.AgM9393.Interop.AgM9393Class' cannot be embedded. Use the applicable interface instead.

To fix this compilation error, right-click on the AgM9393Lib in the project References, and edit the Properties. Change the Embed Interop Types field to "False".



## 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](http://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. Agilent VISA-COM is used by its IVI-Component drivers and requires that Agilent VISA also be installed.

## References

1. Understanding Drivers and Direct I/O, Application Note 1465-3 (Keysight Part Number: 5989-0110EN)
2. Digital Baseband Tuning Technique Speeds Up Testing, by Bill Anklam, Victor Grothen and Doug Olney, Keysight Technologies, Santa Rosa, CA, April 15, 2013, Microwave Journal
3. Accelerate Development of Next Generation 802.11ac Wireless LAN Transmitters-Overview, Application Note (Keysight Part Number: 5990-9872EN)
4. [www.ivifoundation.org](http://www.ivifoundation.org)



This information is subject to change without notice.

© Keysight Technologies 2014 - 2016  
Edition 2.1, May, 2016  
M9393-90007

[www.keysight.com](http://www.keysight.com)