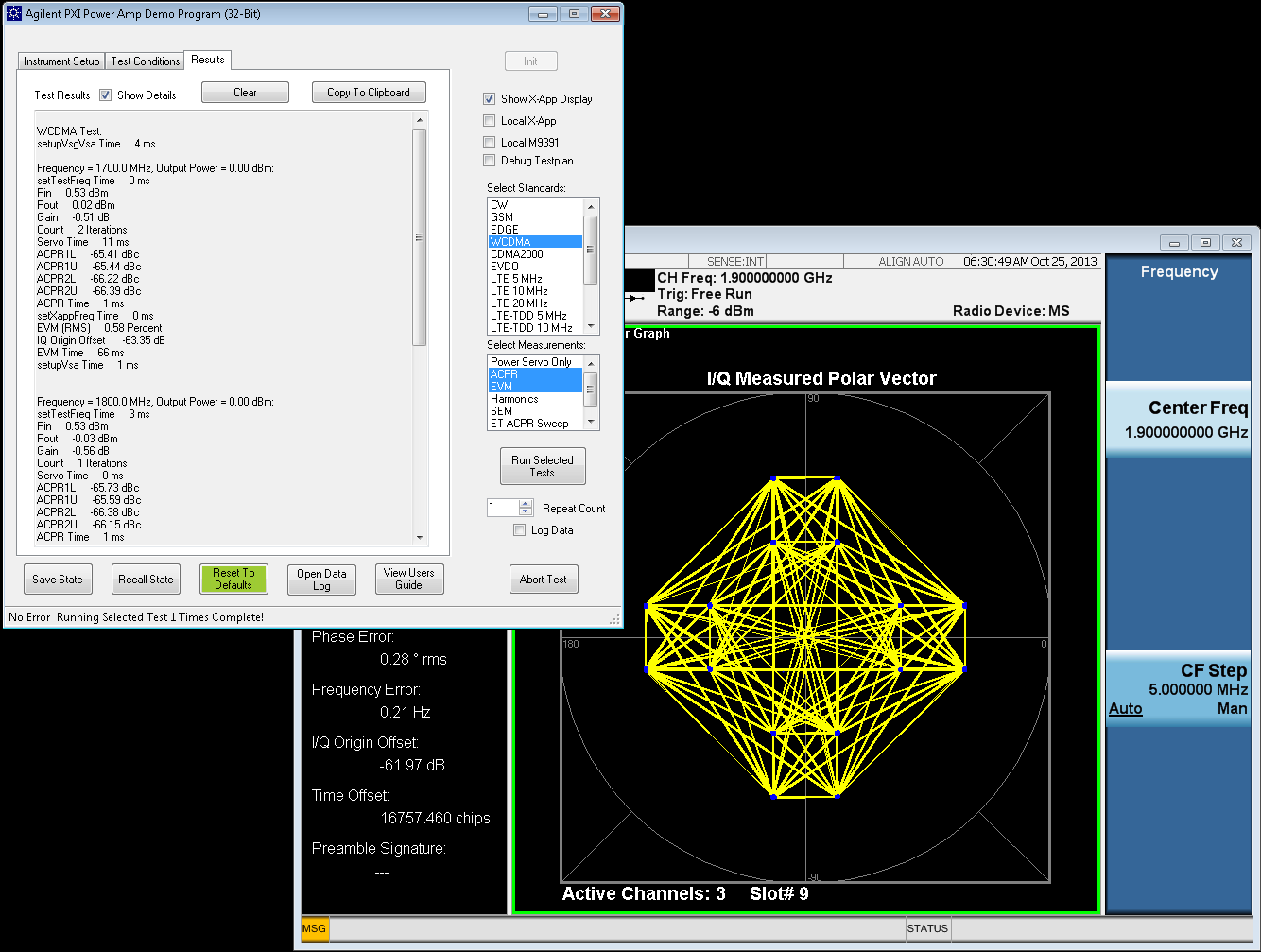
Agilent PXI Power Amp Demo Program

Developer Guide



Contents

[Introduction 3](#_Toc362776162)

[Driver Installation 3](#_Toc362776163)

[M9000 Software Overview 5](#_Toc362776164)

[Agilent PXI Power Amp Demo Program Overview 6](#_Toc362776165)

[PowerAmpTestLib Overview 8](#_Toc362776166)

[PowerAmpDemoGUI Overview 9](#_Toc362776167)

[PowerAmpDemoProgram Overview 10](#_Toc362776168)

[Building the Power Amp Demo Program 11](#_Toc362776169)

[Power Amp Demo Program Details 14](#_Toc362776170)

[PowerAmpDemoGUI Details 14](#_Toc362776171)

[Initialization 14](#_Toc362776172)

[Running Tests: 15](#_Toc362776173)

[Local Mode 17](#_Toc362776174)

[PowerAmpDemoProgram Details 18](#_Toc362776175)

[Initialization 18](#_Toc362776176)

[Required Methods 18](#_Toc362776177)

[Tests 18](#_Toc362776178)

[PowerAmpTestLib Details 21](#_Toc362776179)

[setupVsgVsa 21](#_Toc362776180)

[servoInputPower 22](#_Toc362776181)

[measWlanEvm 23](#_Toc362776182)

[Using the M9000A Software in C# 24](#_Toc362776183)

[Adding the M9000 to the C# Project 24](#_Toc362776184)

[Initializing the M9000 and M9391A 26](#_Toc362776185)

[Visibility of the X-App Window 28](#_Toc362776186)

[Switching Modes In Between the X-Apps and M9391A IVI Driver 29](#_Toc362776187)

[SCPI Compatibility between the X-Series Analyzers and Modular X-Apps 31](#_Toc362776188)

[Using the M9000A Software in LabVIEW 32](#_Toc362776189)

[LabVIEW.exe.config File 32](#_Toc362776190)

[Creating a LabVIEW Project and Config File 32](#_Toc362776191)

[Adding the M9000 Software to LabVIEW VIs 34](#_Toc362776192)

[Direct Control of the M9000 and M9391 in LabVIEW using the .NET Interface 34](#_Toc362776193)

[Using the PowerAmpTestLib in LabVIEW 43](#_Toc362776194)

# Introduction

This document will describe the required procedure to build and modify the Agilent PXI Power Amp Demo Program. Executable versions of this program will be included with all of the M9381A/M9391A Demo Systems. Source code for the PA Demo Program is available from the SMS Factory Application Engineering Team.

This Document will also show the procedure to add the M9000A control program from Modular X-Applications to an existing program. Programming languages contained in this document include:

* C#
* LabVIEW

# Driver Installation

Three IVI driver packages are required to run the PA Demo program. The program will use beta versions of the IVI drivers until the Modular X-App software is released in October 2013. The beta versions of the drivers can be downloaded from [ftp.agilent.com/mdmcust/xapp](ftp://ftp.agilent.com/mdmcust/xapp)

The following are the current versions of the drivers.

* M938x\_Setup-1.3.200.0
* M9000\_Setup-1.0.9.0
* M9391\_Setup-1.1.228.0

The PA Demo program includes a “Read Me File Versions” text file that will document the version of the drivers referenced in the program. If later versions of the drivers are used, it may be required to update the references in the program to the latest version. At this point, we do not expect additional changes to the API for the M9000 software. Functionality may be added to the M908x and M9391 drivers to support MIMO and other functionality to be added in the future.

The M9391 driver must be loaded prior to loading the M9000 driver. Old versions of the M9000 driver should be uninstalled before installing a newer version.

The M9000 package will also install an executable program for starting the X-App software for front panel control and two sample C# programs showing use of the software.

The M9000 Software also requires licenses to run each X-App. Contact the factory for X-App licenses.

# M9000 Software Overview

The M9000 software consists of two primary elements. The first is a version of the X-Applications that use the M9391A VSA. This runs as an application on the PC that also includes the M9391A modules connected via PCI Express. This can be a PXI embedded controller or a PC connected to the PXI mainframe using the M9021A interface card. The second element is a resource manager that allows the M9391A VSA to be used by the X-App software and directly by the M9391A driver without having to close and reopen a session to the hardware. This capability allows the use of the driver for high speed operation of items including ACPR measurement and power servo routines and to use the X-Apps for measurements such as EVM.

In the shared usage mode, IVI driver commands are sent to the M9391A driver, as they would be in other programs that only use the M9391 without the M9000 software and SCPI commands are sent to the X-Apps software as they would to the X-Apps running in a MXA or PXA.

There are new commands to communicate with the resource manager to switch between modes. At a high level, the resource manager “locks” the M9391A hardware to either the M9391A driver or to the X-App software. To switch the hardware from one mode to the other, the hardware must be “unlocked” from the first program before the second program can use it. The X-App software automatically checks out the hardware before each measurement and then checks the hardware back after the measurement is complete. For the M9391A driver, you must send a command to the M9000 software to check out and check back in. This process will be shown in the example code.

# Agilent PXI Power Amp Demo Program Overview

Executable versions of the PA Demo Program are installed on the M9381A/M9391A demo system embedded controllers. The folder containing the executable programs can be copied to other computers to run, as long as the target computer has the required drivers installed and licenses for the X-Apps, if they are to be used.

The PA Demo Program can be configured from the front panel to allow use for testing and collecting data under a large number of conditions. The goal is that many customer evaluations can be completed just using the executable version of the program. A user’s guide and instructional video will also be included on the demo system.

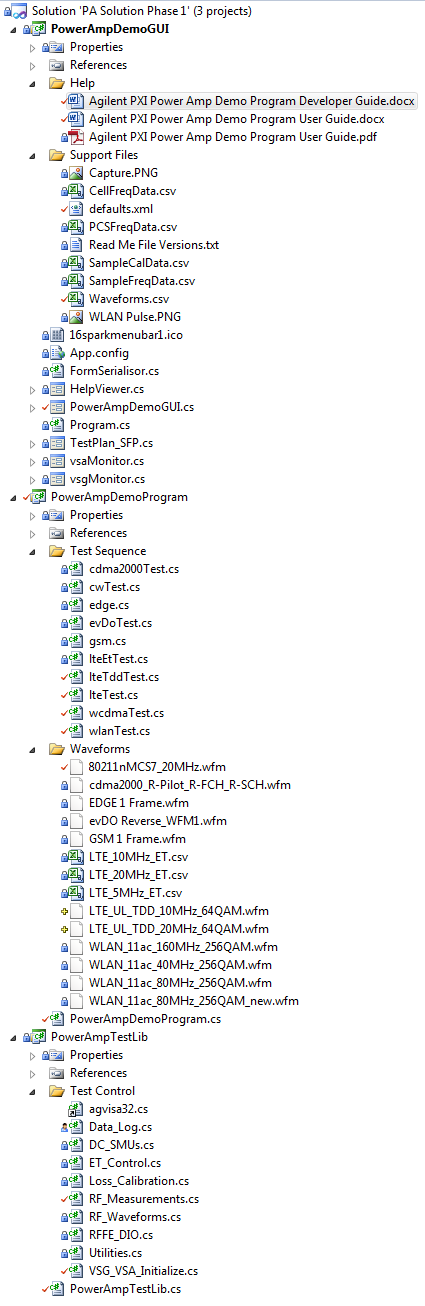
In some cases there may be a need to modify the PA Demo program or provide the source code to a customer. A customer may want to include some of the code from the demo program into their own test software or may want to use the code as an example for best practices for various measurements.

This section of the document will provide instructions on how to build the solution and an overview of the solution structure.

The Demo program is a Visual Studio 2010 C# solution containing 3 projects:

* PowerAmpDemoGUI: The user interface for the demo program
* PowerAmpDemoProgram: The test sequence for the specific device
* PowerAmpTestLib: All of the instrument control, measurement and initialization routines.

The following screen capture shows the files currently included in the solution:



## PowerAmpTestLib Overview

The PowerAmpTestLib project includes the drivers for the M9381A, M9391A and M9000 X-App software. The driver sessions and most of the properties in the TestLib are public so that they can be modified in either the TestProgram or GUI.

The project includes the following files:

* PowerAmpTestLib.cs: Definition of the member variables used in the project and the constructor for the class.
* VSG\_VSA\_Initialize.cs: Includes the startup, closing and state control for the VSG, VSA and X-App software. Also contains code for configuring the triggers for the VSG. VSA and X-Apps and routing the triggers on the PXI backplane.
* RF\_Measurements.cs: Includes methods for setup and measurement of power amplifier tests using the AgM938x and AgM9391 IVI drivers, and the X-App software. Some of these routines are common across all standards, such as the Input Power Servo routine. Some of the routines are specific to a specific standard, such as the ACPR routine for LTE.
* Data\_Log.cs: Methods for displaying the data for a single run in the results window of the GUI and for saving all of the values to a .CSV file
* Loss\_Calibration.cs: Methods for applying a correction factor to the input signal level and measured output levels of the DUT and a procedure for calibrating system losses with a power sensor
* Utilites.cs: Utility methods, such as delay and units conversion
* agVisa32.cs: Link to Agilent I/O library VISA routines used to send SCPI commands to the X-Apps

This project can be included in a solution to provide instrument control of the PXI VSG, VSA and the X-Apps. It can also be used as a reference for using the instruments to make power amp measurements. The constructor for the class has three input parameters that are used to create the size of the data log variables.

## PowerAmpDemoGUI Overview

The powerAmpDemoGUI project provides the user interfaces for the PA Demo Program. The use of the program is covered in the Agilent PXI Power Amp Program Users Guide document and will not be covered in this document. In general, the GUI provides buttons that execute methods in the TestProgram project to open and close the instruments and run the tests. Additional controls allow the use set properties in the TestProgram and TestLib projects to configure how the tests will be run. As well as show the results after the measurements have been run. All of the controls in the GUI include tool tip help text that is shown when the user’s mouse hovers over the control.

The project includes the following files.

* PowerAmpDemoGUI.cs: A windows form that is the main user interface for the PA Demo Program.
* vsaMonitor.cs: A windows form that allows viewing and changing setting for the AgM9391 IVI driver and also allows making measurements in the IQ, power, FFT and spectrum acquisition modes. The power and FFT modes are used in power and ACPR measurement and the spectrum mode is used for some harmonic measurements. The IQ mode is used in some envelope tracking measurements to look at AM and PM conversion.
* vsgMonitor.cs: A windows form that allows viewing and changing settings for the AgM938x IVI driver.
* HelpViewer.cs: a windows form that is used to view the user’s guide from inside the program.
* FormSerialiser.cs: A C# code file that is used to save the state of the main GUI form controls to an XML file and also to load the controls from the contents of an XML file. When the PA Demo Program exits, the state of the GUI is stored into a file named lastState.xml. When the program is launched, the state of the GUI is read from the same file.
* Help Files: MS Word versions of this document and the user’s guide. Also a PDF version of the user’s guide. The PDF version is copied to the program folder when the project is built.
* Support Files: .CSV template files for user defined frequency and calibration and the .XML files described above. These files are all copied to the program folder when the project tis built.

## PowerAmpDemoProgram Overview

The PowerAmpDemoProgram contains several methods that are called by the GUI project to configure and run tests. It also includes a number of methods that consist of the test sequence for each of the standards supported in the program.

The file PowerAmpTestProgram.cs includes the constructor for the class and the methods that interface with the GUI. The constructor creates an instance of the PowerAmpTestLib when it is run. The methods called from the GUI include:

* ConfigTestLists: The GUI includes a three listBox controls for displaying the list of X-Apps, the supported standards and the available measurements. This method populates those controls based on the measurements that are included in the test program.
* InitializeInstruments: Performs the initialization routines in the TestLib for the PXI instruments and X-App software and loads all of the required waveforms into the M9381A VSG
* runSelectedTests: Runs a subset of the standards and measurements based on the current values in the standards and measurements listBox controls in the GUI.
* CloseInstruments: performs the close instrument method in the TestLib
* runPowerCalibration: sets up the conditions and runs the power calibration method in the test library

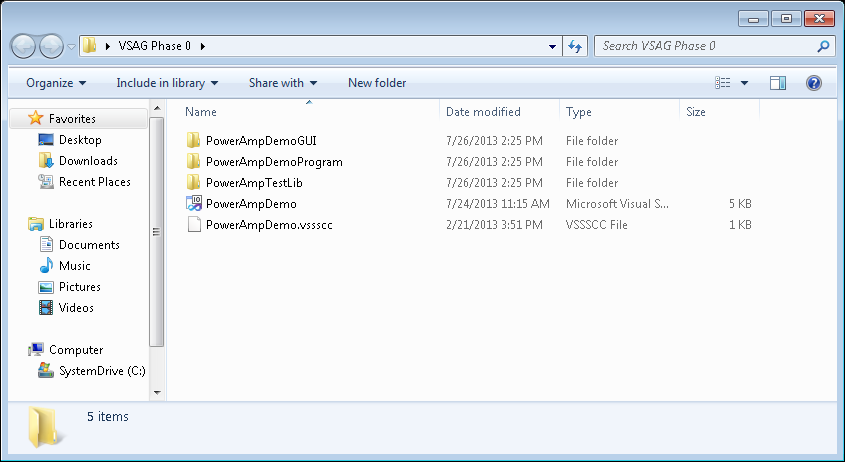
The additional files in this project include:

* Test Sequences: The test sequence method for each standard supported in the Test Program. The method for each standard is included in a separate code file with the name of the standard. Some of the test sequence methods, such as LTE and WLAN have a parameter to specify the bandwidth of the standard to be tested.
* Waveforms: In most cases, the example waveforms installed with the M938x driver are used in the PA Demo Program. Some standards required waveforms that were not included with the examples. These waveform files are included in the project. The waveforms are automatically copied to the program folder when the project is built

# Building the Power Amp Demo Program

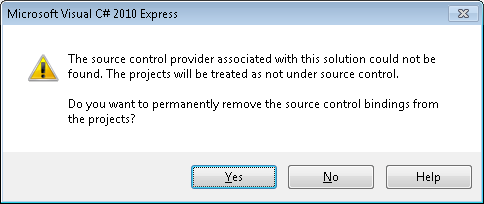
The Power Amp Demo Program should be built using Visual Studio 2010. The M9000 software requires .NET version 4.0 and thus the 2010 version of Visual Studio. The program can be built using the standard version of Visual Studio or any of the advanced versions. It can also be built using the Express Version, with minor modifications to the post build events.

The source code for the project will be provided in a zip file. After unzipping, the following folder structure will be available:



To load the program into Visual Studio, double click the solution file or select File | Open Project or Solution from the Visual Studio Menu.

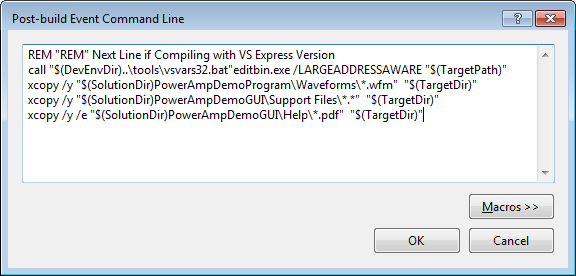
While the solution is loading, you may see the following dialog concerning source control bindings:



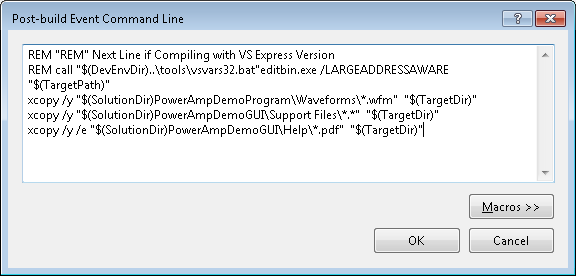
You can either say yes to remove the bindings or just ignore this message. The program is under source control in SMS and the bindings are not removed before making the source code available.

If the correct versions of the IVI drivers are loaded, the program should compile and build successfully. There should be no warnings or errors. If there are errors or warnings about the driver references, insure that the required versions of the drivers are installed or remove references to the drivers and add new references to the installed versions. The references for the drivers are includes in all three projects.

The GUI project includes several post build events that allow the x86 version of the program to use large address space and copy the waveform and support files to the output folder.



The “vsvars.bat” post build event is used to increase the available memory in the x86 build of the program. The larger memory space is required to allow loading multiple X-Apps. The tools required to run this script are not included with the Express version of the Visual Studio. The project can still be built by editing the post build event command line and inserting “REM” before the work “copy”, as shown below:



The x86 version of the program will still run without this step, but the number of X-Apps that can be loaded will be limited to 2-4, depending on which apps are selected.

# Power Amp Demo Program Details

## PowerAmpDemoGUI Details

### Initialization

The Power Amp Demo GUI creates a PowerAmpDemoProgram object and also includes objects for the local panels for the VSG and VSA and a form to view the help file. When the PowerAmpDemoProgram Object is created, it will generate an instance of the PowerAmpTestLib.

The The GUI will call the following methods in the PowerAmpDempProgram Object:

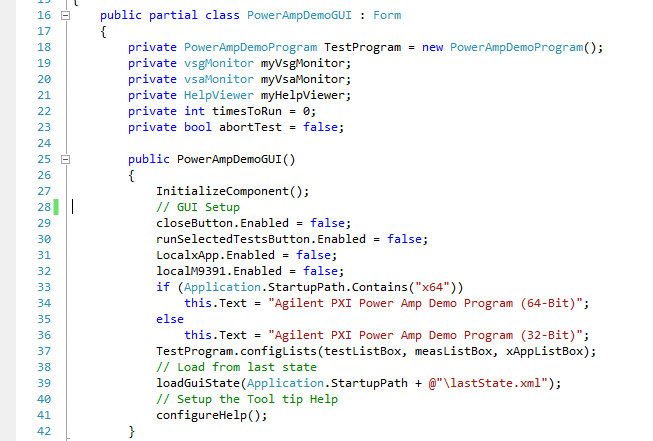
* ConfigTestLists: Called from the GUI contructor, sets the list of values to be shown in the X-App, Standards and Measurements lists. The TestProgram should set these to the values supported in that program.
* InitializeInstruments: Called from the Init Button event handler, opens all of the Instruments used in the test program. Accepts a parameter with the list of X-Apps to be loaded, if the useXApp property is set to true
* CloseInstruments: Called from the Close button event handler and the form closing event handler, closes all of the driver sessions and terminates the X-App software. It is required to call this from the form closing event handler. If the program exits without closing the drivers, the PC will need to be rebooted before the instruments can be used again.
* readCalData: Called from the Load Frequency Data File event handler: The path and file name of the frequency data is passed. The test frequency arrays are contained in the test program
* initFreqFreqs: Called from the Default Frequency event handler, resets the test frequencies to the default values defined in the test program.
* runSelectedTests: Called from the Run Selected Tests Button event handler, runs the tests selected in the standards and measurement lists. Takes parameters that are lists of strings for the standards names and measurement names.
* runPowerCalibration: Called from the Run Calibration button event handler. Passes the VISA resource for the power meter/sensor and the file name for the calibration data.

The GUI will call several methods in the PowerAmpTestLib:

* setXAppMode: Called in the Local XApp event handler, this is used to put the X-Apps into local and remote modes based on the value of the Local X-App check box
* OpenDataLog: Called in the Run Tests event handler, it will open the data file with the parameter of the data file path and name. At this time, the path of the data file is hard coded to C:\Temp\logFile.csv.
* createHeader: Called from the Run Tests event handler with a parameter of the repeat count. The repeat count is used to correctly format the statistics formulas that are included in the log file.
* CloseDataLog: Called at the end of the Run tests event handler to close the data log.

Several properties in the TestLib, such as the power servo settings, instrument resource strings, and data log parameters will be set to values based on GUI controls. These controls all use a common event handler, setTestProgramValues.

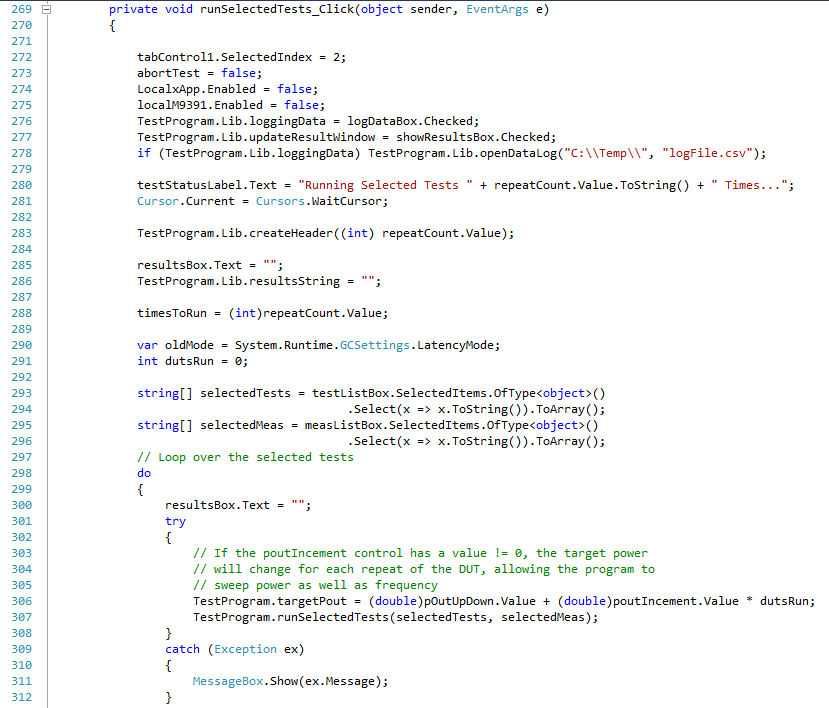
The enabled state of the open, close, run test buttons and the local check boxes are controlled in GUI to match the correct state based on the condition of the program. For example, the open button is enabled on startup and is disabled when the instruments are initialized. All of the methods that control the state of the instruments will set these values to the correct state. Lines 27-42 below are run when the program starts. Lines 33-38 set the text of the windows form to include 32 or 64 bit. Line 39 reads the state of the controls from a file. The method loadGuiState reads the values from the xml file. If there are file names in the frequency or calibration data file fields, it will load those file and set the test program properties to the correct values. Line 41 loads the tool tip help text for each control.

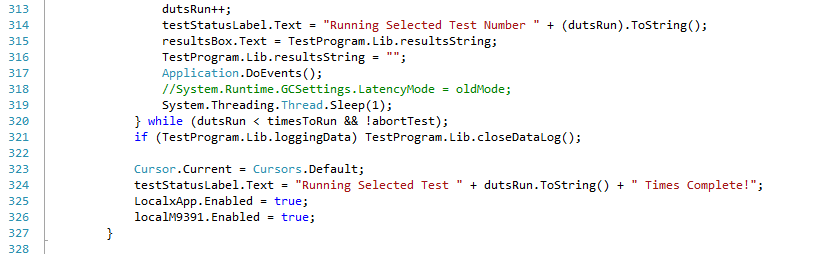


### Running Tests:

The event handler for the run selected tests button calls the method in the Test Program to run the tests. It also configures the data log and checks if the abort button has been pressed.

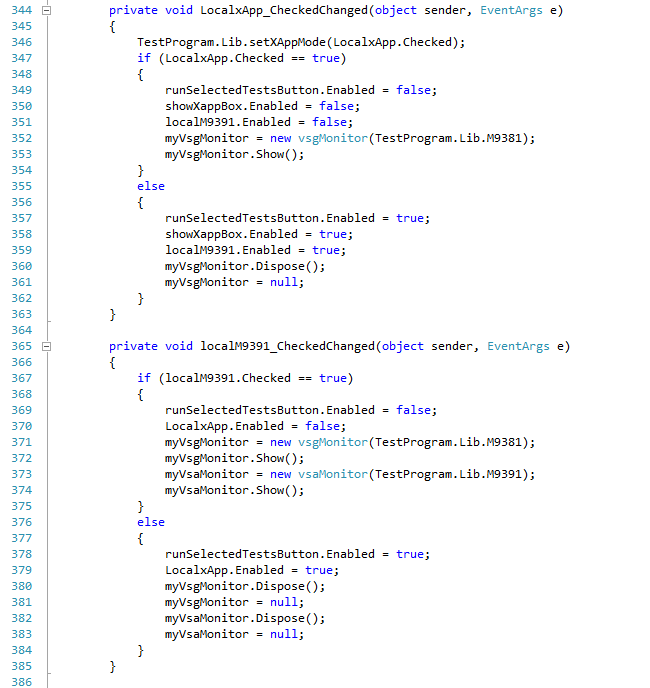
Line 273 sets the Abort test property to false. This property is set to true when the Abort button is pressed. Lines 274-275 disable setting the X-Apps or VSA drivers into local mode. Since the local mode forms use the driver, they would be able to change the setup of the instruments while the tests are running. To prevent this, local mode is not allowed while the tests are running. Also, the tests can not be run when either the X-Apps or the VSA are in local mode. Lines 301-312 run the tests and catch any exceptions handled in the program. Line 306 adjusts the Target power for the servo loop, if required. Line 317 allows other events to be handled. This is specifically added to check if the Abort button has been pressed. At line 320, the loop will terminate if the repeat count is met or if the user aborted the tests.





### Local Mode

The local mode event handlers will create an object for the VSG and VSA monitor panels, passing in the driver reference from the TestLib. These objects are disposed when local mode exits. The Run Tests Button is disabled while local mode is active to prevent local modes changes to the drivers while tests are running.



## PowerAmpDemoProgram Details

### Initialization

The PowerAmpDemoProgram Object will create an instance of the PowerAmpTestLib in the constructor. The constructor for the TestLib takes parameter to size the data log variables. This allows all of the data acquired in one complete run of the test program to be available in the data log at the end of the run.

### Required Methods

The structure of the PS demo program allows for different test programs to be used with the same GUI, or to be used independently of the GUI. The following methods are called from PowerAmpDemoGUI and must be included in the Test Program:

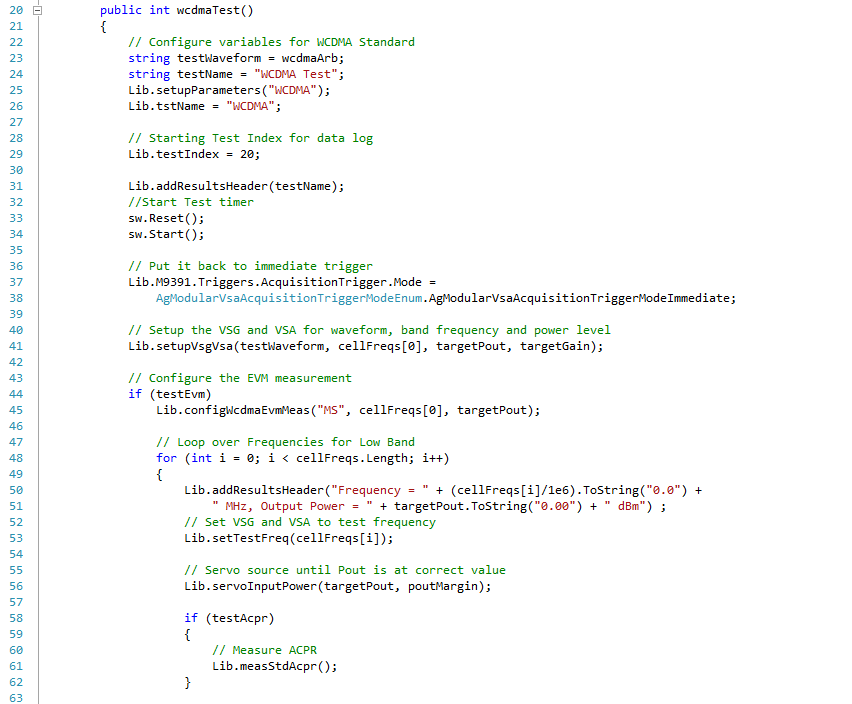
* ConfigTestLists: Populates the lists of X-App names, Standards Names and Measurement Names. These should match the tests performed in the test program.
* InitializeInstruments: Opens all of the drivers and load all of the required waveforms into the VSG.
* CloseInstruments: Closes all of the drivers.
* runSelectedTests: Runs the test methods based on the list of standards and measurements passed from the GUI.
* readFreqData: Load the test frequencies from a CSV file
* initTestFreqs: Sets the test frequencies to the default values

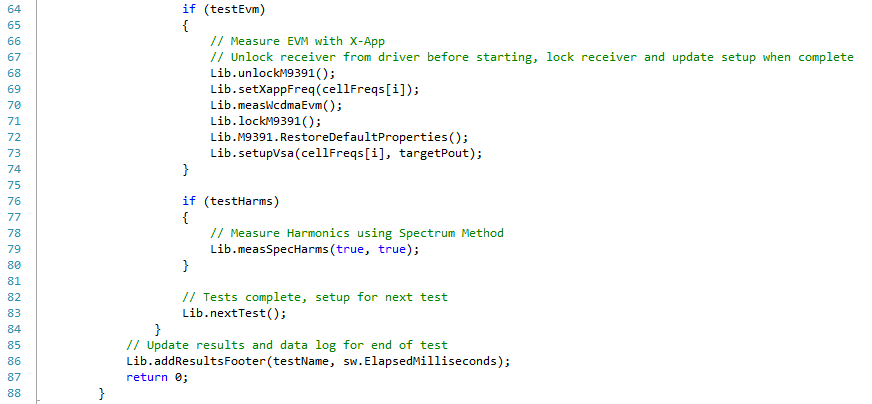
### Tests

The tests for each standard are contained in a separate code file. All of these files share a common structure, with the basic flow as shown below:

* Parameter Setup: Set the name of the ARB waveform to be used and configure a number of basic parameters in the VSA, such as bandwidths, filter shapes and sample rates. This is done by passing the name of the standard to the “setupParameters” method in the TestLib. LTE and WLAN have an parameter for the bandwidth, which is used in this section for the required setup
* Start the Test Timer
* Setup the VSG and VSA to the initial conditions for the test
* If EVM is being measured, setup the X-App to the correct mode and configure the EVM measurement for that mode
* Loop over the list if test frequencies
  + Set the VSG and VSA to the correct frequency
  + Servo to the correct output power
  + For each measurement, if enabled, make the measurement. If the measurement uses the X-Apps, follows these steps:
    - Unlock the VSA from the driver
    - Set the X-App Frequency
    - Make the measurement
    - Lock the VSA to the driver
    - Execute the “Reset to Default Properties” method of the VSA Driver
    - Reset the VSA frequency, power level and trigger mode
* Stop the Timer
* Record the total test time in the data log

The following is an example of the above process:





## PowerAmpTestLib Details

The primary function of the PowerAmpTestLib is to use the IVI driver for the VSG and VSA and the SCPI interface for the X-Apps to setup and make the required measurements for the testing of power amplifiers. In addition to this basic function, the TestLib also contains methods to apply corrections for the losses at the DUT input and output. The DUT input loss is applied to the VSG output levels. The DUT output loss is applied to the VSA reference level and the power measurements. The TestLib also includes a data logging function that will put the measured results into a text string that is displayed in the GUI and into a CSV file that can be used with Microsoft Excel or other spreadsheet programs.

After reviewing the structure of a test sequence shown above, it makes sense to look at some of the procedures in the TestLib in more detail. Some of the methods called in the above example will be shown in detail to describe the operation of the test lib.

### setupVsgVsa



This method configures the VSG and VSA to perform a power servo to the target output power on a device with the target gain. The getInputAtten and getOuptutAtten methods are used to compensate for the loss at the DUT input and output.

Line 284 calculates the RF level of the source so that the value of the member variable servoHeadRoom of baseband power offset will provide the ideal input level to the DUT.

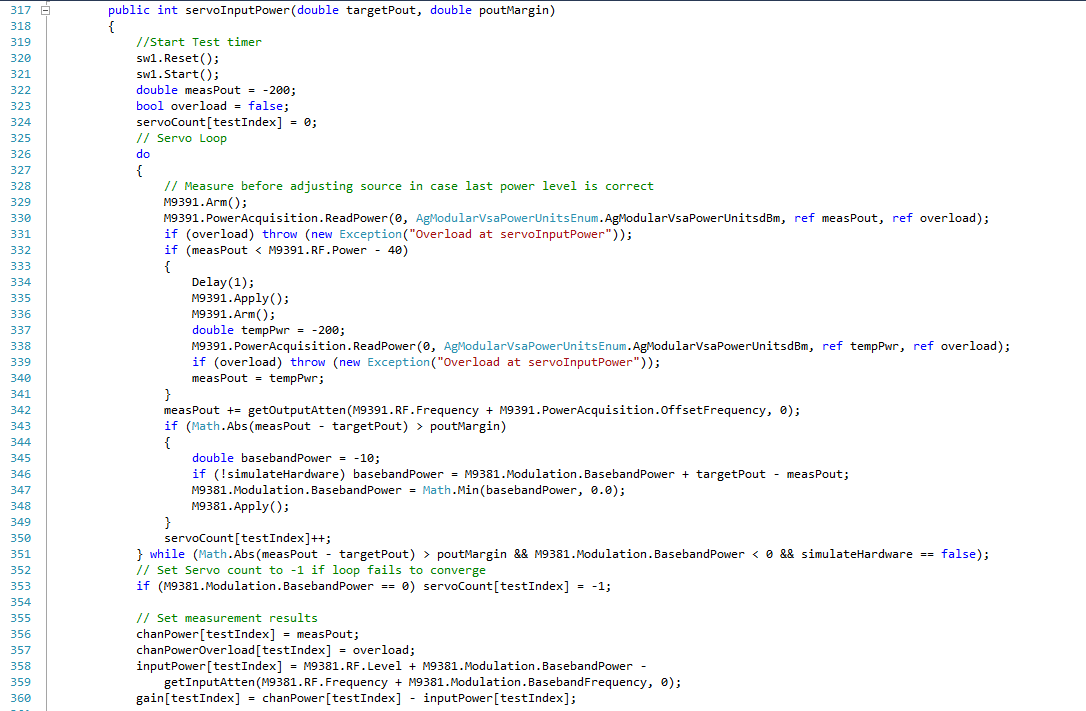
The VSA reference level is set based on the target output power level of the DUT, the output loss and a margin level to keep the reference level above the expected output level of the DUT. This value should be large enough to prevent overloads in the RF measurements, but if it is too large it will decrease the dynamic range of the measurements. This value has been determined empirically for each standard and is set in the “configParameters” method.

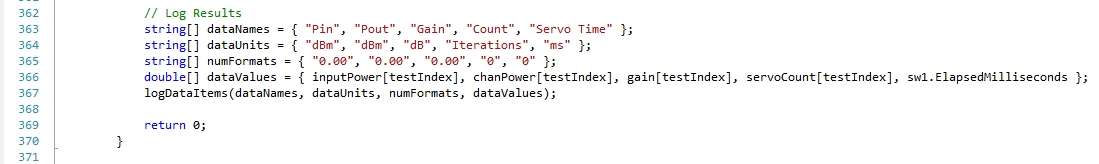
The peak to average ratio of the ARB waveform is used to set the peak to average value of the VSA. This parameter allows you to use the average power level of the DUT output when setting the reference level.

Release 1.1 of the M9391 driver added the FFT Acquisition Mode. In this mode, block of IQ data are run through a hardware FFT in the signal processing chain of the M9214A digitizer. This new mode can be used to speed up power servos and ACPR measurements. The setupVsgVsa method will setup either the power mode or the FFT mode depending on the value of the member variable fftServo.

Release 1.3 of the M938x driver added a new method to playback a subset of a loaded waveform, supporting the use of “short waveforms” that allow external triggering with minimum test times. The setupVsgVsa method now supports playback of the full waveform, a shortened waveform or a waveform sequence. Waveform sequences are used for WLAN testing to allow changing the duty cycle of the input signal without changing the input waveform.

### servoInputPower



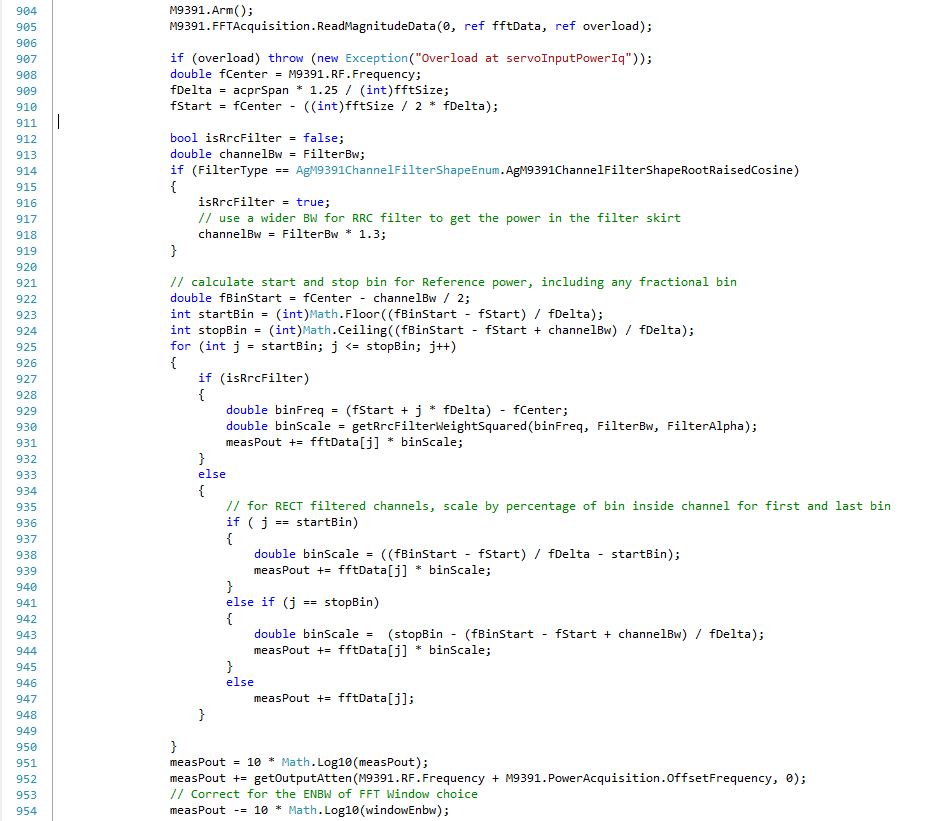


Lines 317-351 perform the input servo loop. Since the setup routine set the VSG level to the ideal level, the procuedre measures the output power first and then adjusts the power level if requried. When this routine is run in a loop, the value that worked for the previous iteration will be tsted first. This is done to mimimize the number of times that the source must be changed. If the servo loop is unable to converge due to a DUT with low gain, the input level will be set to the maximum and the servo loop count will be set to -1.

Line 339 shows the normal procedure for overloaded measurements with the reciever. An exception is thrown with the name of the test. This exception should be handled at a higher level in the program. In this demo program the excepctions are handled in the GUI method that runs a partial or full test. Thus, an exception will end the test for a DUT. In some cases it may be preferrable to handle the exceptions in the test seuqence method, such as the WLAN test shown above.

Lines 362-367 copy the measurement results to the data log. The TestLib has a common data logging method that is used in all tests that add data to the log. The data logging method formats both the text string displayed in the results window and the data/hearders for the CSV file.

In addition to the above input power servo method there is a new method that performs the servo using the FFT acquistion. When using the FFT acquistion mode, the power in the channel is caluclated by summing thevoltage in the FFT bins included in the channel, applying a rectangualr or RRC filter, as requried. The speed of the servo loop in power and FFT acquistions is basically the same. The FFT acquisiton allows calculation of the ACPR values using the last measurement from the servo loop and will produce the lowest overall test times. There is an issue in the 1.1 release of the M9391A IVI driver that prevents correct external triggering of the FFT acquisition that will result in a higher standard deviation in power measurement calculated from the FFT. With this release, the apporach with the best repeatability is to use the power acquisition for the servo loop and then a single FFT measurement for the ACPR measurement. The fastest test times will be from using the FFT measurement for both the power servo and the ACPR calculations. The user can select which servo mode to use from the RF Settings tab in the GUI. The following code shows the approach for determining the channel power from the FFT acquisition results:



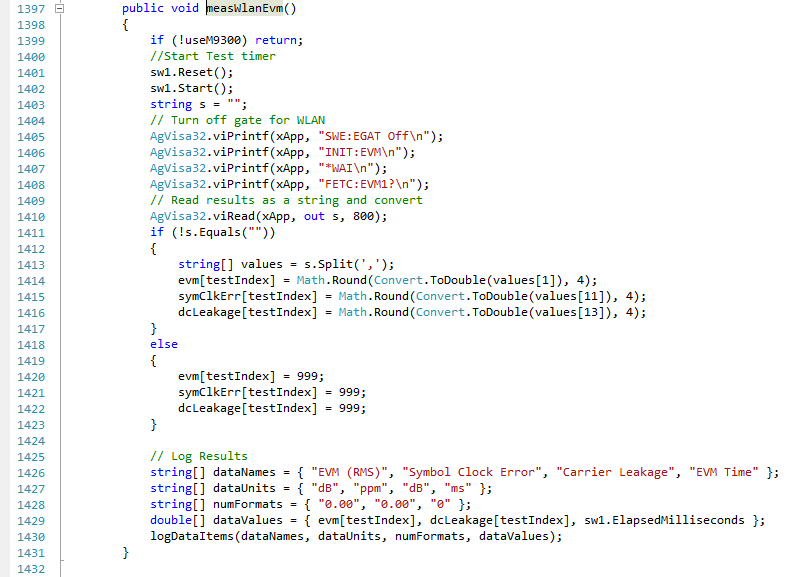
Lines 904-905 acquire the FFT data. Lines 908-910 calculate the frequencies for the FFT bins. Lines 912-924 deterimne which FFT bins should be included in the channel power calculation. For channels with RRC filter, a wider bandwidht is used and then the RRC filter shape is applied to the bins at the edges of the channels.

Lines 925-950 loop over the bins in the channel and add up all of the voltages for these bins. This routine supports rectangular and RRC filtered channels. For RRC filtered channels, the voltage of a bin is multiplied by the filter weight. Inside the filter, the weight will be 1. Outside the filter, it will be 0. In the transition region it will be between 0 and 1. For rectangular filtered channels, the first and last bins are multiplied by the percentage of the bin that is inside the channel.

Lines 951-954 convert the voltages to power in dBm, add the output attenuation and compensate for the effective noise bandwidht of the FFT window function, which was determined in the FFT Acquisition setup in setupVsgVsa.

This procedure is repeated for each adjacent channel in the ACPR measurements that use the FFT acquisiton mode.

### measWlanEvm



This is an example of a method that sends SCPI commands to the X-App and then parses the result string into the measured values. It uses the same approach for logging the results data.

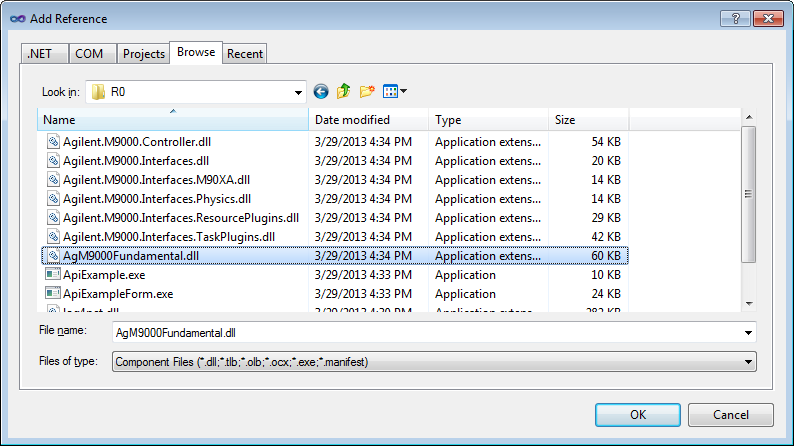
# Using the M9000A Software in C#

## Adding the M9000 to the C# Project

With the final release of the M9000 software, the M9000 Fundamental driver and the M90XA IVI driver will be used. There are several steps to add the software to the project:

* Add a reference to the M9000 Fundamental
* Add a reference to the M90XA IVI driver
* Add a reference to the Agilent.M9000.Interfaces library
* Use the “Agilent.M9000” namespace
* Use the “Agilent. M90XA.Interop” namespace;
* Use the “Agilent.M9000.Interfaces” namespace;
* Create an “app.config” file or add an entry to and existing app.config file
* Set the Target .NET Framework to 4.0

To add the reference, right click on the references folder for the project in the Visual Studio Solution Explorer and select “Add Reference.” Select browse and browse to the folder: C:\Program Files (x86)\Agilent\M9000\Core\R0. Then select AgM9000Fundalmental.dll. Do the same for Agilent.M9000.Interfaces. The AgM90xALib IVI driver can be added from the COM tab in the add reference dialog.



To use the namespace, add the following line to any source files that will access the M9000 software:

using Agilent.AgM90XA.Interop;

using Agilent.M9000;

using Agilent.M9000.Interfaces;

The app.config file is required to force the correct versions of the .Net runtime library to be used. If an app.config file does not exist in the current solution, right click on the startup project name and select add new item. From the list of Visual C# items, select “Application Configuration File.” Copy the following text into the App.config file:

<?xml version="1.0"?>

<configuration>

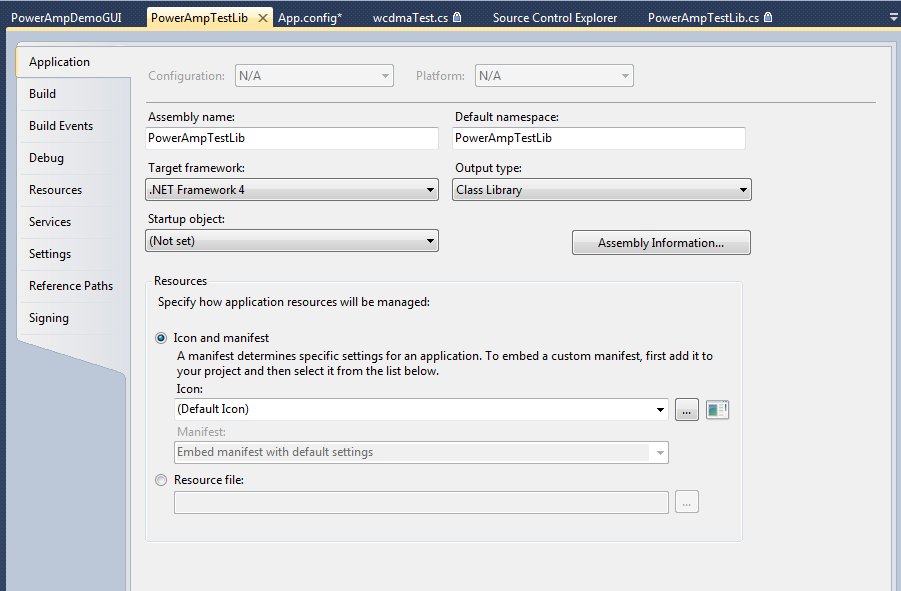
<startup useLegacyV2RuntimeActivationPolicy="true">

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0" />

</startup>

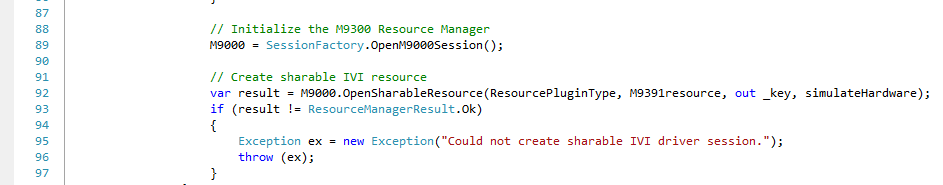
</configuration>

To set the Target .NET framework, right click on the project name in the Solution Explorer and select “Properties.” In the properties dialog, set the target framework to “.NET Framework 4.” Do not use the “.NET Framework 4 Client Profile” setting, as it will not work.



## Initializing the M9000 and M9391A

Once the above steps are completed, the first step in using the code is to initialize the M9000 software and M9391A driver. The procedure for initializing the M9391A driver is different when using it with the M9000 software compared to using the M9391A by itself. The Sample program uses a Boolean to determine if the M9000 software will be used. In one case the M9391A is started in the normal manner and in the other case, the procedure for use with the M9000 is used. The following example shows the procedure. A description of the process follows. The following is a screen capture to allow showing the line number for the explanation. The actual code will follow, allowing copy and paste into a program.



Lines 89 starts the M9000 resource manager. Line 92 is creating the M9391A resource that will be shared between the driver and the X-App. The M9391resource variable is the resource name and must be an IVI configuration store saved connection. Unlike using the driver in the normal mode, you cannot supply a list of PXI addresses in this command.

The following is the actual code for initializing the resource manager that you can copy and paste into a program.

// Initialize the M9300 Resource Manager

M9000 = SessionFactory.OpenM9000Session();

// Create sharable IVI resource

var result = M9000.OpenSharableResource(ResourcePluginType, M9391resource, out \_key, simulateHardware);

if (result != ResourceManagerResult.Ok)

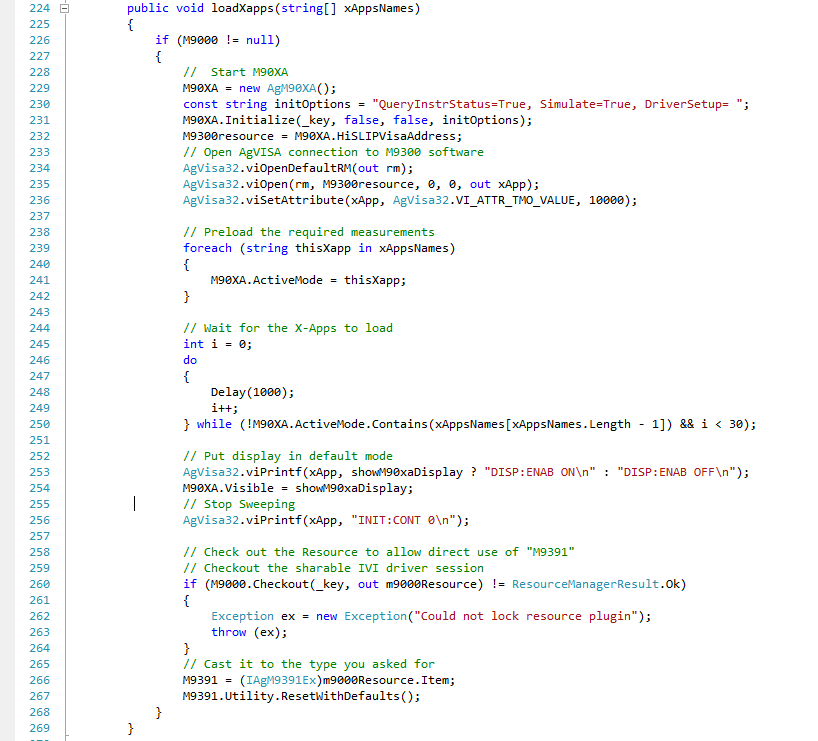
{

Exception ex = new Exception("Could not create sharable IVI driver session.");

throw (ex);

}

The following code will Start the XApps software using the M90XA driver and load a list of X-Apps passed into the method.



Line 231 starts the X-App software. Lines 241 set the active X-App, for each X-App in the supplied list. Starting multiple X-apps will increase the startup time, but will “preload” the required X-Apps. After the active mode is set to each of the X-Apps, loading all of the measurements, the program waits until the X-Apps return that the active mode is the last X-App loaded. After all of the X-Apps have loaded, the display is put into the correct state and continuous sweep is disabled.

Line 232 returns a string that is the resource used to send SCPI commands to the X-Apps. In this example program, we have linked in the AgVIsa32 library for SCPI communications. To use this approach, a reference to the AgVisa32.cs file, included with the Agilent I/O library installation needs to be added to the project

After the X-App initialization is complete, the M9391 resource is checked out from the resource manager in line 183 and the checked out resource is casted to the type of the M9391A driver in line 189. After this command, the M9391 driver can be used as normal. A reset is sent to hard M9391A driver in line 190. Note that we are using IAgModularVsa, a new interface that will make porting to future Agilent PXI VSA drivers easier. This will need to be included as a reference in your program before using.

The following is the text of the initialization procedure that can be copied into a program:

public void loadXapps(string[] xAppsNames)

{

if (M9000 != null)

{

// Start M90XA

M90XA = new AgM90XA();

const string initOptions = "QueryInstrStatus=True, Simulate=True, DriverSetup= ";

M90XA.Initialize(\_key, false, false, initOptions);

M9300resource = M90XA.HiSLIPVisaAddress;

// Open AgVISA connection to M9300 software

AgVisa32.viOpenDefaultRM(out rm);

AgVisa32.viOpen(rm, M9300resource, 0, 0, out xApp);

AgVisa32.viSetAttribute(xApp, AgVisa32.VI\_ATTR\_TMO\_VALUE, 10000);

// Preload the required measurements

foreach (string thisXapp in xAppsNames)

{

M90XA.ActiveMode = thisXapp;

}

// Wait for the X-Apps to load

int i = 0;

do

{

Delay(1000);

i++;

} while (!M90XA.ActiveMode.Contains(xAppsNames[xAppsNames.Length - 1]) && i < 30);

// Put display in default mode

AgVisa32.viPrintf(xApp, showM90xaDisplay ? "DISP:ENAB ON\n" : "DISP:ENAB OFF\n");

M90XA.Visible = showM90xaDisplay;

// Stop Sweeping

AgVisa32.viPrintf(xApp, "INIT:CONT 0\n");

// Check out the Resource to allow direct use of "M9391"

// Checkout the sharable IVI driver session

if (M9000.Checkout(\_key, out m9000Resource) != ResourceManagerResult.Ok)

{

Exception ex = new Exception("Could not lock resource plugin");

throw (ex);

}

// Cast it to the type you asked for

M9391 = (IAgM9391Ex)m9000Resource.Item;

M9391.Utility.ResetWithDefaults();

}

}

## Visibility of the X-App Window

There are two components to the X-App visibility that are used in the above initialization procedure:

* The “Display Enable” SCPI command shown on line 179 of the example will determine if the UI is updated as SCPI commands are sent to the program and measurements are performed. This behaves the same as on the MXA/PXA. To view the results of the tests turn the display on. For optimal test times, turn the display off
* The value of the property:  
  M90XA.Visible  
  Shown on line 180 of the sample program determines if the window containing the X-Apps will be visible or minimized.

In the example program, the Display enable and visible values are set from a check box on the user interface for the program. This allows you to leave the X-App visible, aiding with demonstration and debug and to minimize with the display off, allowing for fastest execution time. There is also a command on the User interface to “Local” the X-Apps. This allows the X-App display to be turned on and off for debugging. More details on this function will follow in a later section.

## Switching Modes In Between the X-Apps and M9391A IVI Driver

Using the initialization procedure described above allows use of both the X-Apps and the direct control of the M9391A IVI driver. This is desirable because the M9391A IVI driver allows high speed measurements for items such as input power servo loops and ACPR measurements. The X-Apps are required for measurements such as EVM and SEM.

Three different use models should be considered:

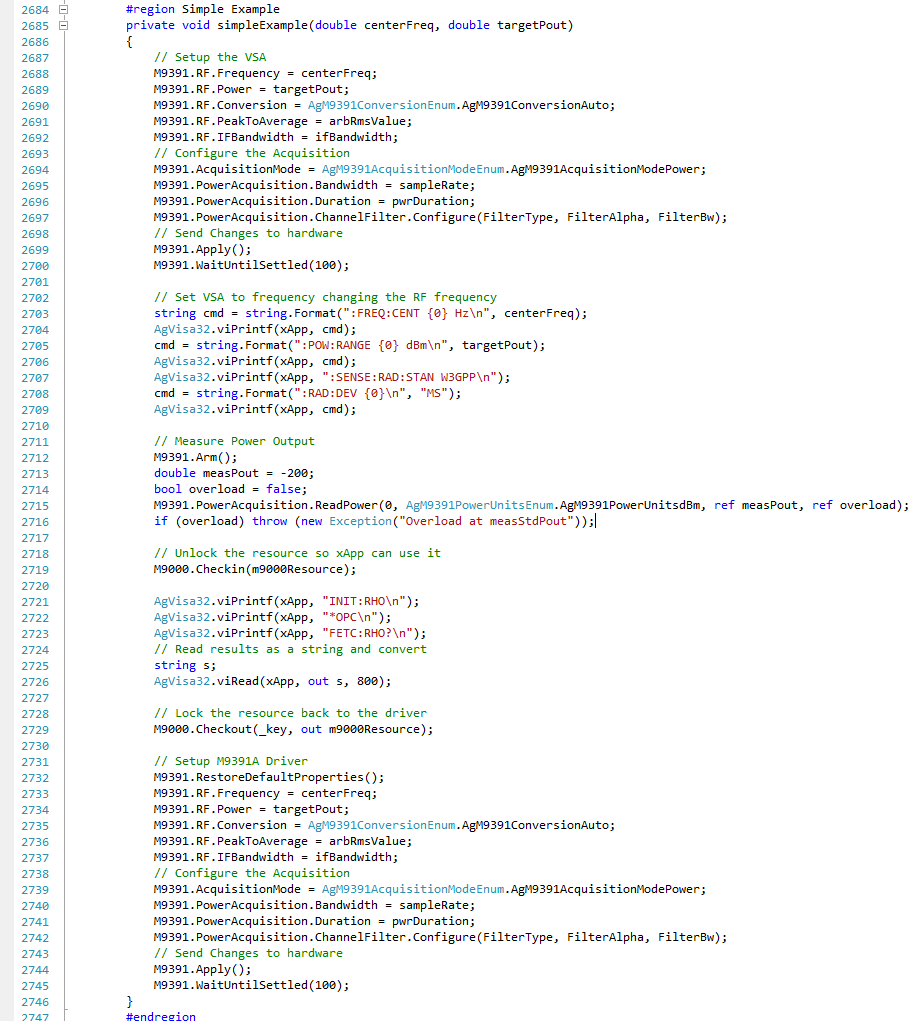
* X-Apps Only
* Shared use of X-Apps and IVI Driver
* IVI Driver only

For the X-App only case, the above initialization procedure should be used up to line 180. After this point, the program can simply send SCPI commands to the M9000Resource and the program. For the IVI driver only mode, the M9000 software is not required and the M9391A IVI driver should be initialized in the normal manner.

For the shared case, the above initialization procedure should be used in full. At the end of the above procedure, the M9391A hardware is checked out to the IVI driver. At this point, any valid command can be sent to the IVI driver and IVI driver acquisitions can be made, as in the IVI driver only mode.

To switch to the X-App mode, the M9391A resource must be checked back into the resource manager and then SCPI commands can be sent to the X-App. After the X-App measurements are complete, the M9391A hardware must be checked out from the resource manager before driver can be used again.

The following is a simple code example that could be performed directly after the initialization procedure shown above:



Lines 2687-2700 use the M9391A driver to configure a power measurement. This code is exactly the same as the case without the M9000 software. Lines 2702-2709 setup a WCDMA EVM measurement to be made with the X-App. These commands are exactly like you would send to the MXA/PXA with the exception of the “POW:RANGE” command on line 2705. More details will be provided on this point later in this document.

Lines 2711-2716 make the power measurement using the M9391A driver. Line 2719 unlocks the M9391A hardware, allowing the X-App to use the hardware. Lines 2721-2726 use the X-App and perform an EVM measurement. Line 2729 checks the M9391A hardware back out from the resource manager to allow further use with the M9391A driver.

There are several additional important points to highlight from the above example:

* The “setup” commands were sent to the X-App while the IVI driver still had the resource checked out to the M9391 IVI driver. The setup commands are cached in the X-App code until a measurement is actually performed. So, in this example, these commands were sent to the X-App program, but no changes were sent to the hardware until the measurement was initiated.
* There was no command for the X-App to check out or in the M9391A hardware. The X-App program automatically checks out the resource before performing a measurement and automatically checks the resource back in after the measurement is complete.
* **The “setup” of the M9391A driver needed to be updated after the X-App used the M9391A hardware. The state of the hardware is not maintained from the time the resource is checked in to when the resource is checked back out. Since there is no way to know exactly how the hardware was used in the X-App, the safest way to deal with this is to use the M9391A driver “restoreDefaultParameters” command and then repeat the complete setup. This command will have a minimal impact on test time. In the above example, we have just repeated the setup that was done at the beginning of the example. If this code was run in a loop, you could also do the X-App measurements last, and then setup the next test condition at the top of the loop.**

## SCPI Compatibility between the X-Series Analyzers and Modular X-Apps

Since the hardware block diagrams between the X-Series Analyzer and the M9391A PXI VSA are significantly different, there will need to be some differences in the SCPI commands between the two platforms. Since the Modular X-App software is still in development, the full extent of these differences in not yet known. There has been one new SCPI command created for the Modular X-Apps at this time:  
**:POWer:RANGe xx dBm**This command is used to set the input range of the M9391A VSA. In the X-series Analyzers there are commands to set the value of the mechanical and electrical attenuators. In the current state, the range command has no effect on the X-series analyzers and the attenuator commands have no effect on the M9391A VSA. Thus, a compatible program could be generated by using both the range and the attenuator commands.

Refer to the M90XA Getting Started Guide for a complete list of the SCPI difference between the Modular X-Apps and the X-Apps running on an xSA.

# Using the M9000A Software in LabVIEW

This section will describe the procedures to use the M9000 software with LabVIEW 2011. Other versions of LabVIEW have not been tested at this time. The following steps are required to use the M9000 Software:

* Create a LabVIEW.exe.config file allow the use of .NET framework 4.0
* Create a LabVIEW project for the VIs that will use the M9000 software
* Create the LabVIEW project configuration file to allow use of the Agilent License manager
* Add one or more VIs to the project that will use the M9000 software

## LabVIEW.exe.config File

By default, LabVIEW 2011 will use the .NET framework 3.5. The M9000 code requires the .NET framework 4.0. To create the config file, copy the following text into a text file and save it with the name LabVIEW.exe.config in the folder that contains the LabVIEW.exe file. The typical location for this file is C:\Program Files (x86)\National Instruments\LabVIEW 2011

<configuration>

<startup useLegacyV2RuntimeActivationPolicy="true">

<supportedRuntime version="v4.0.30319"/>

</startup>

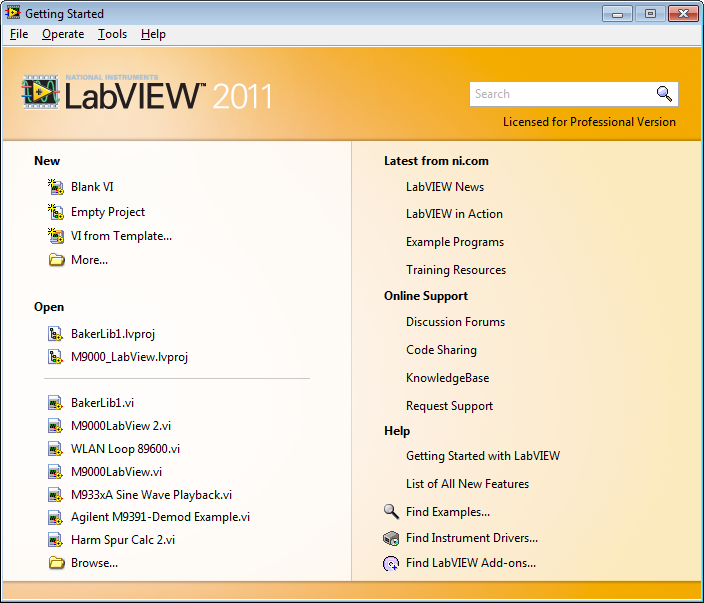
</configuration>

The above process is described in detail in the following article from NI: <http://zone.ni.com/reference/en-XX/help/371361J-01/lvconcepts/config_net_app/>

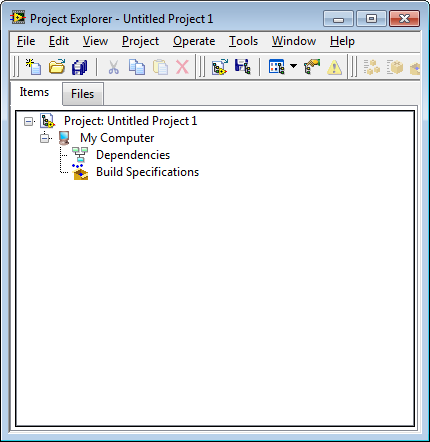
## Creating a LabVIEW Project and Config File

The M9000 software requires access to the Agilent license manager through a .NET assembly. This requires configuration when used in LabVIEW. NI suggests that all projects that reference .NET assemblies be included in a LabVIEW project. The following article from NI describes this requirement in detail: <http://zone.ni.com/reference/en-XX/help/371361J-01/lvconcepts/config_net_app/>

From the LabVIEW getting started screen, select New: Empty Project



The following blank project display will be shown:



Save the project. For example, use the name M9000 ExampleProject.lvproj

Using a text editor add the following text to a file and save to the same location as the LabVIEW project with the same name, adding .config to the end of the complete File Name. With the example name shown above, the file name would be M9000 ExampleProject.lvproj.config

<?xml version ="1.0"?>

<configuration>

<configSections>

<section name="xSA.Services" type="System.Configuration.SingleTagSectionHandler, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>

</configSections>

<xSA.Services Service1="Agilent License Service"/>

<appSettings>

<add key="ServiceWatch.Interval" value="500"/>

<add key="ServiceWatch.NumerOfTimes" value="600"/>

</appSettings>

</configuration>

If you are adding the M9000 software to an existing LabVIEW project, create the project config file as shown above or add this text into an existing project config file.

## Adding the M9000 Software to LabVIEW VIs

Two approaches for using the M9000 software in LabVIEW will be presented.

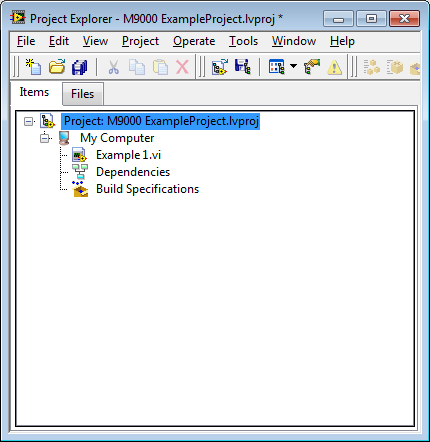
The first approach is to use the >NET interface in LabVIEW to directly control the M9000 software and the M9391A IVI driver. At this time, use the M9391A LabVIEW driver is not supported. However, the full capability of the M9391A driver can be accessed using the .NET interop assembly.

The second approach uses the LabVIEW .Net interface to control the PowerAmpTestLib described above. This has the advantage of providing a higher level interface to use in LabVIEW, but has the disadvantage of having some of the program in LabVIEW and some of the program in C#.

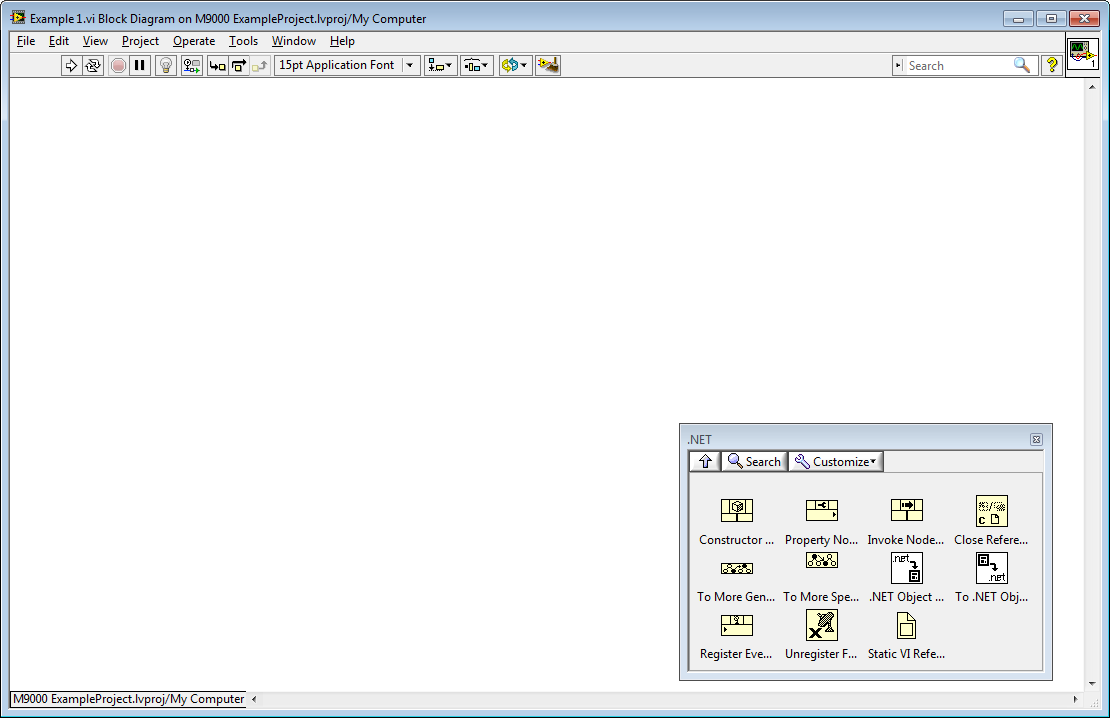
Review the following two sections to determine the best path for your situation.

### Direct Control of the M9000 and M9391 in LabVIEW using the .NET Interface

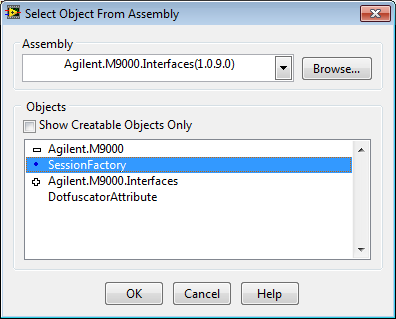
In the above create project, select File | New VI to add a new VI to the project. Save the VI. Note that the project dialog should now include the new VI:



The connectivity -> .NET palette will be used to add the required .NET items to the VI.

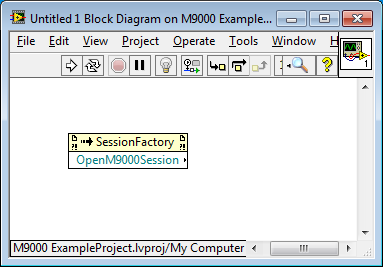


Add a .NET Invoke Node to the VI. Right click on the node and then select from the menu “Select Class | .NET | Browse.” Browse to the path C:\Program Files (x86)\Agilent\M9000\Core\R0. Then select Agilent.M9000.Interfaces.dll and press OK.

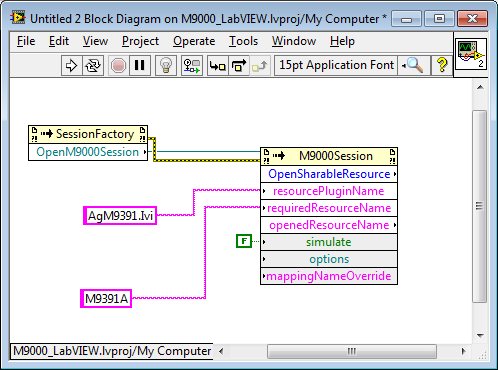


From the list of Objects in the above dialog select “SessionFactory” and press OK.

Right click on the invoke node and from the menu select “Select Method | OpenM9000Session” The VI should look like the following:

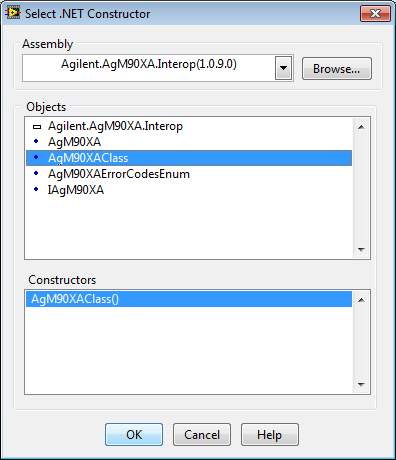


Add another Invoke Node to the block diagram, connect the reference input to the “OpenM9000Session” reference output of the first invoke node and select the method “OpenSharableResource”. Wire the rest of the terminal as shown below. Controls or constants may be used for the IVI Config name and the simulate Boolean.

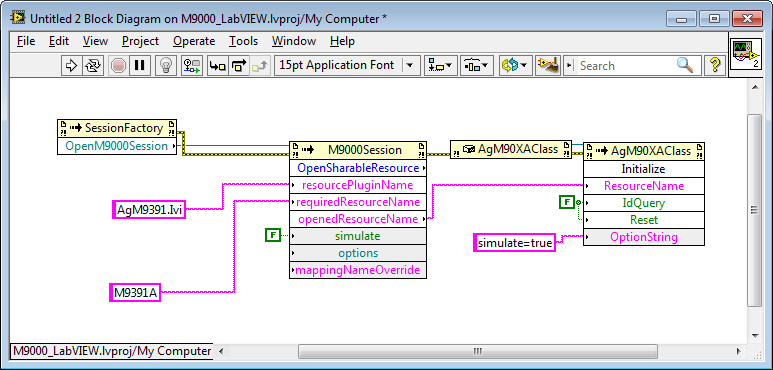


Add a .Net Constructor node to the block diagram. Right click and choose select constructor from the menu. Browse to Agilent.AgM90XA.Interop.dll in the IVI bin folder:   
C:\Program Files (x86)\IVI Foundation\IVI\Bin\Primary Interop Assemblies

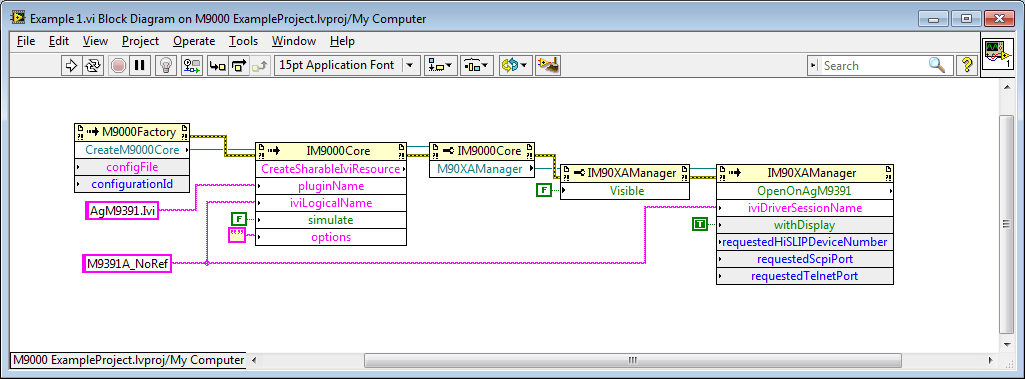
Select the AgM90XAClass from the list of objects.



Add an invoke node and wire as shown in the following diagram. After the reference inputs have been connected, you can select the property names and method names by clicking on the node.

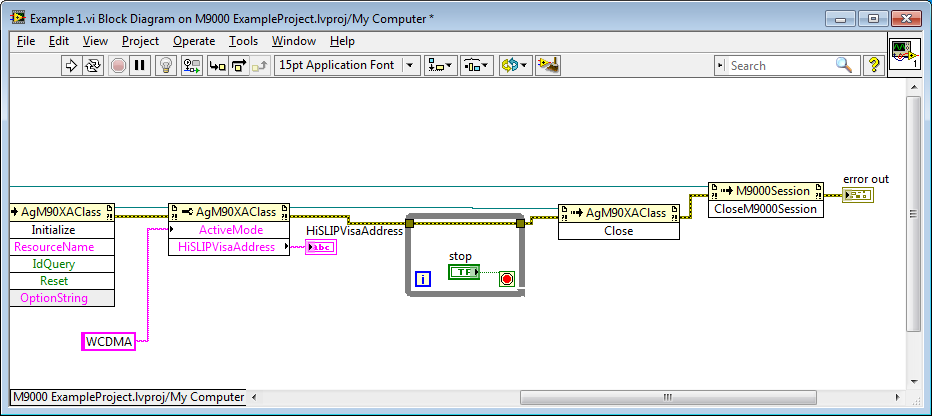


This part of the program will create the resource manager and open the X-App container with a blank screen, as no X-apps have been loaded at this point.



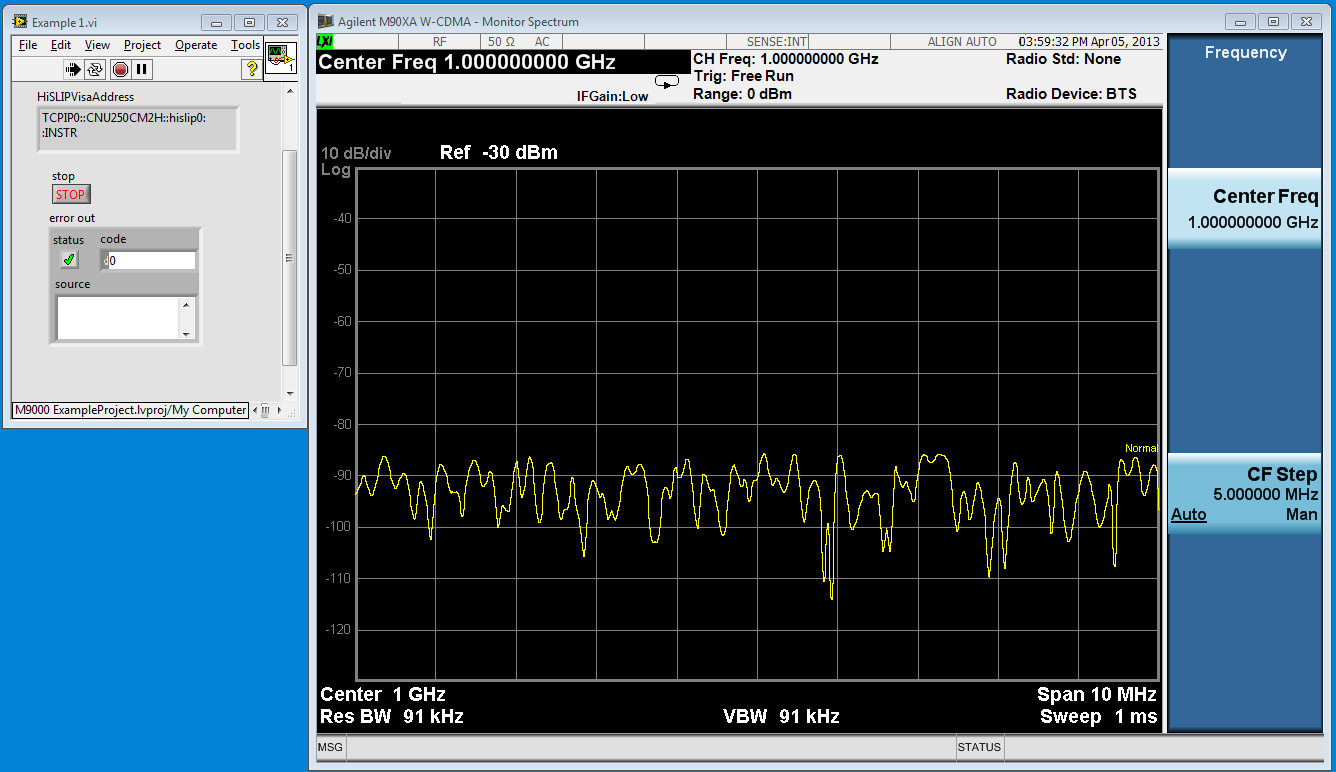
Add a property node, connect the reference terminal and select the “ActiveMode” property. Connect a string constant or control to the input and enter the name of one of the X-Apps. This will cause that X-App to load on startup. Several X-apps can be loaded at startup, but at least one needs to be loaded. On the bottom of the property node, pull down the bottom edge to add a second property. Use this property to read the HiSlipVisaAddress. This address can be used to send SCPI commands to the X-Apps.

Next, add a while loop, wiring the error connector through the loop. Add a control to stop the loop. This will allow you to interactively control the X-app until the stop button is pressed. Add two additional invoke nodes after the while loop to close the Xapp Class and dispose of the M90000Session. Note the connections of the reference terminals, as shown in the following diagram.



Add an error indicator to the end of the program, save and run.

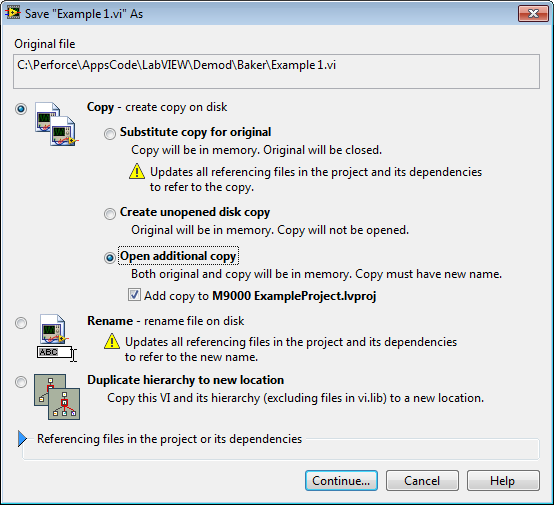
As the program runs, the X-App window will open and can be used interactively until the stop button in the LabVIEW front panel is pressed.



After the stop button is pressed the X-App GUI will disappear and the error indicator should show no error.

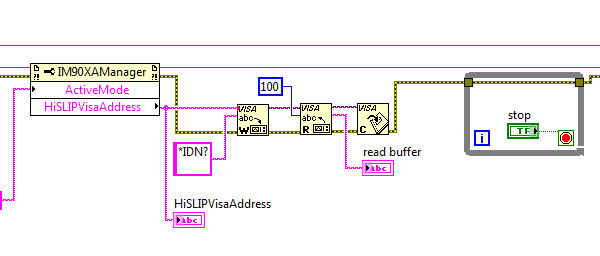
Once the X-app is initialized, the two primary tasks will be to send SCPI commands to the X-App and using the IVI driver for the M9391A for power and other measurements. LabVIEW supports several methods for sending SCPI commands. The use of the VISA read and write locks will be shown here. For the M9391A IVI driver, the resource must be checked out and controlled by the .NET interop assembly, in much the same manner as in C#.

Save the Example 1.vi used above. Then select “File | Save As…” from the LabVIEW menu. In the save as dialog, select “open an additional copy” and check the “add to project” box:

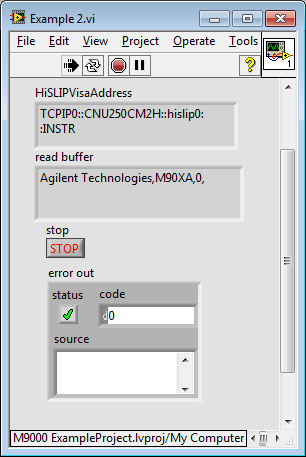


Press continue, and name the new program Example 2.vi.

Example 2 will add sending SCPI commands to the previous example. Add the VISA write, read and close objects to the program before the while loop as shown below. Note the synchronous mode is set to synchronous and the SCPI command need to be terminated with a new line character. The VISA resource name is the output of the HiSlipVisaAddress property. Add an indicator to the output of the read command.



Run the program and observe that the IDN string is displayed in the read buffer indicator.

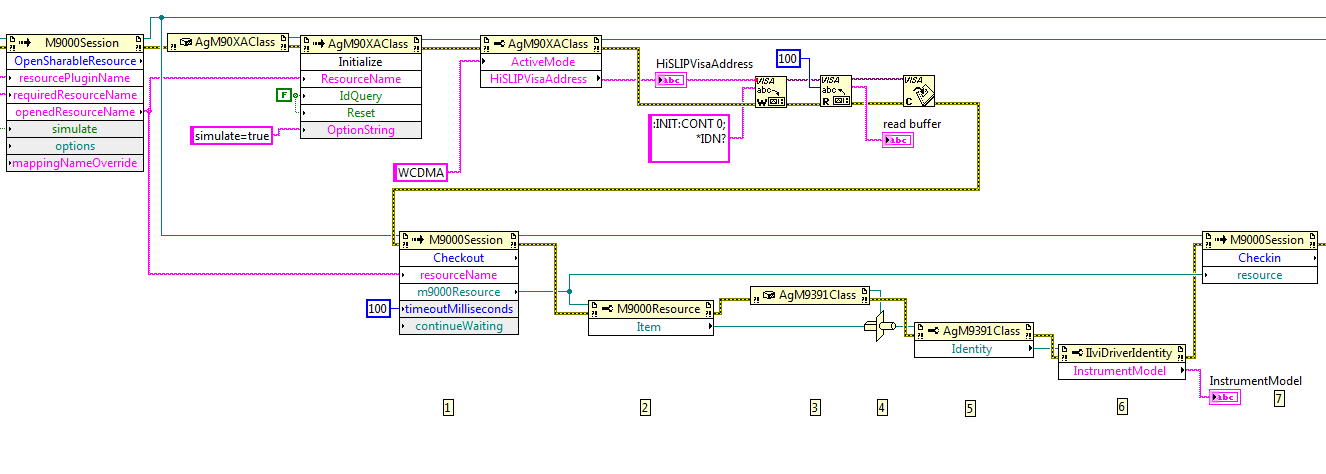


Save Example 2 and then “Save As” an additional copy as “Example 3.” The final point to cover is using the M9391A IVI driver. The procedure is basically the same as in the C# examples:

* Check out the hardware resource
* Cast it to the M9391A IVI driver
* Use the Driver to control the hardware
* Check the hardware back in to allow use by the X-Apps

Example 3 will use the driver to read the model number of the receiver before sending the SCPI commands to the X-App.

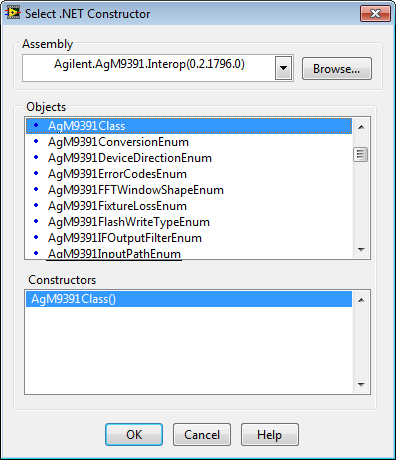
The following block diagram adds 6 new steps for using the IVI driver that will be explained below. It also highlights one of the difficulties with using the .NET interface in LabVIEW, the program block diagrams can grow quite large. In some cases, multiple .NET invoke and property nodes will be “stacked vertically” to make better use of the screen space. Another approach is use “multiple rows” of .NET items which flow from left to right and top to bottom. Also, the .NET nodes tend to have several connections from node to node for reference terminals, further complicating the block diagram. In any of these approaches, make sure to use the error connections to ensure the program will execute in the correct order.



Explanation of new Items:

Explanation of new Items:

* 1: Invoke node on M9000Session to check out the shared hardware resource, requires input parameters of the key name, which is an output of the OpenSharableResource.
* 2: Property node on M9000Resource to get the item.
* 3: Constructor of type AgM9391 Class. Add a constructor from the .NET palette and then select the AgM9391 Class, as shown in the following diagram:

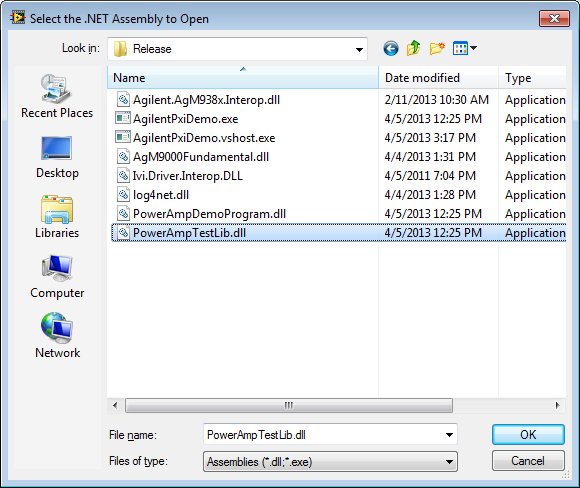


* 4: Cast the output of the property node in step 3 to the type of the AgM9391 driver from step 4
* 5, 6: Use the M9391A IVI driver using the .NET invoke and property nodes. In this example the identity and instrument model properties are read from the driver.
* 7: Invoke node on the M9000Session to check in the shared resource to allow the X-app to use the hardware. The input parameter, m9000Resource from the “checkout” is used in this method.

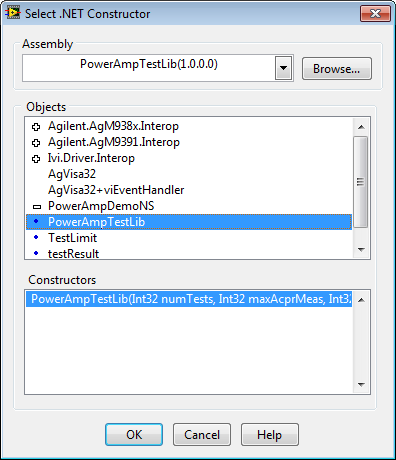
### Using the PowerAmpTestLib in LabVIEW

The PowerAmpTestLib described above is developed in C# as a library, and as such, can be accessed in LabVIEW using the .NET interface. To use this library, the LabVIEW program will follow the examples shown in the PowerAmpTestProgram C# example above.

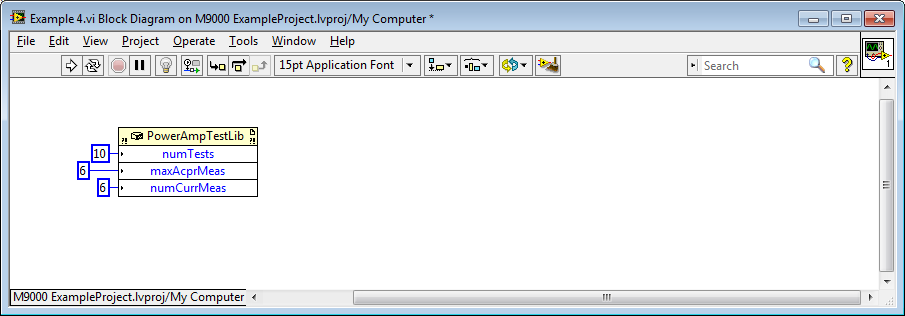
To add the PowerAmpTestLib to an LabVIEW, start with a new VI in the above project and add a .NET constructor, and then browse to the PowerAmpTestLib.dll as shown below:



Next, select the PowerAmpTestLib object, as shown below:



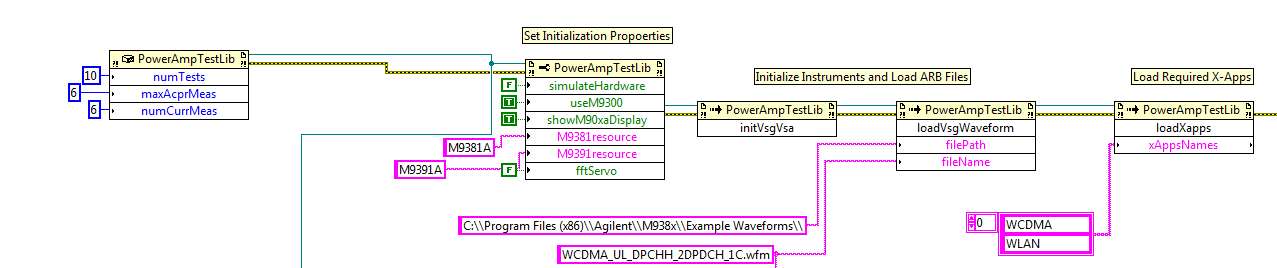
The constructor will now be shown on the VI block diagram. Provide constants for the size of the data arrays to complete the constructor:



The initialization process is completed with the following steps:

* Use .NET property node to set properties to select simulation of hardware, use of M9000 software and visibility of the X-App displays.
* Use Invoke Node to Initialize the VSG, VSA and M9000 software
* Use Invoke Node to Load any ARB files that will be used
* Use Invoke Node to preload any X-Apps that will be used. At least one X-App needs to be loaded

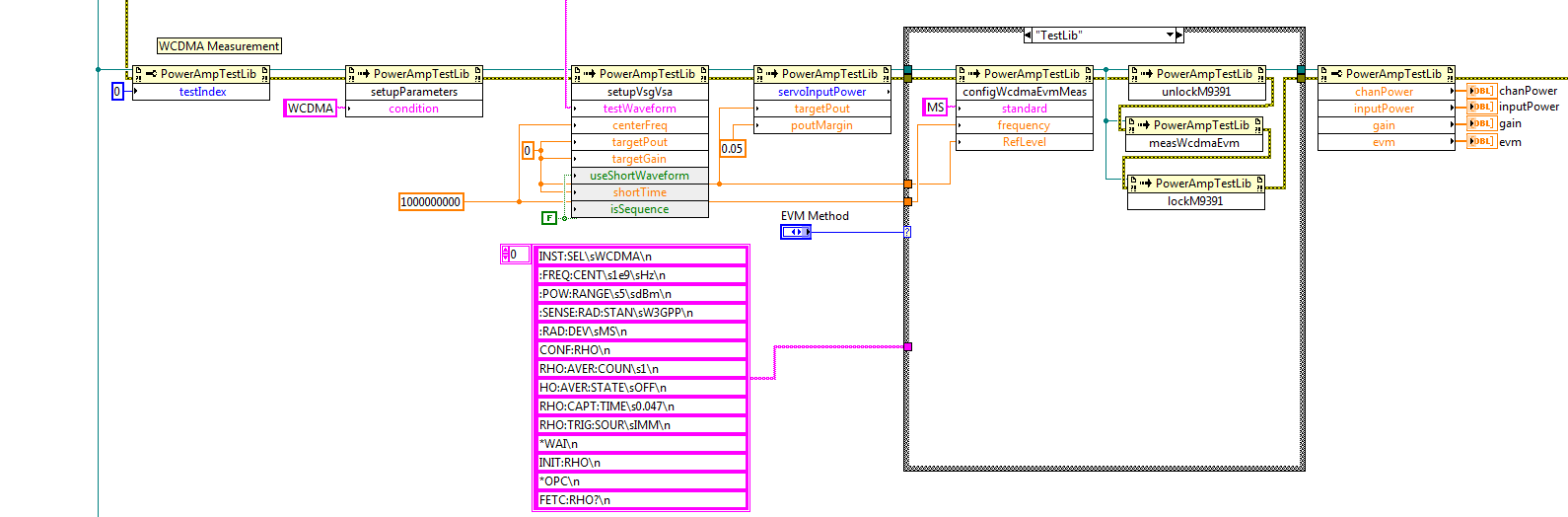
These steps are shown in the following screen capture:



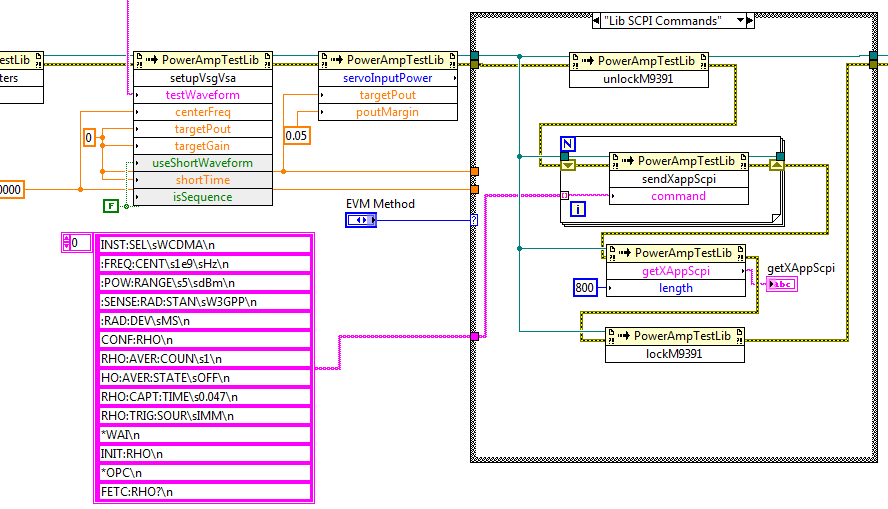
The tests will be performed in a manner similar to the test c# files described above:

* Set testIndex parameter for data log
* Call setParameters to setup for the specific standard
* Call setupVsgVsa to initialize the test conditions
* Set input level or servo to the correct output level
* Make any required measurements. For M9391A IVI driver based measurements, the powerAmpTestLib methods can be called. For X-App measurements, the powerAmpTestLib methods can also be called, or SCPI commands can be sent directly to the X-Apps from LabVIEW. The PowerAmpTestLib includes a method to send arbitrary SCPI commands, or the LabVEIW VISA commands can be used. The following example will show all three methods for making X-App based measurements:
* Read the measured values from the data log variables

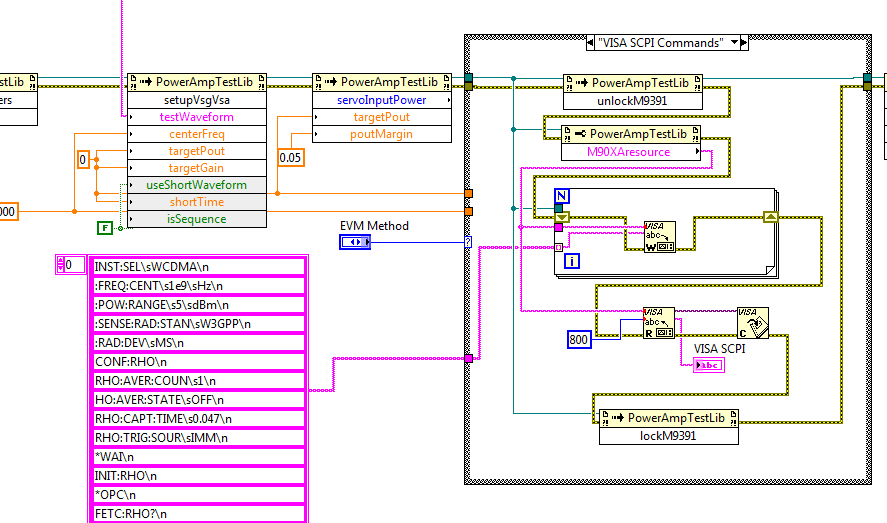
The following example shows these steps added to the VI:



In the above example, the EVM measurement is performed using the method from the PowerAmpTestLib. Note the list of SCPI commands is not used in this example. The following example shows using the PowerAmpTestLib methods for sending arbitrary SCPI commands and reading back results:



The final example shows using the LabVIEW VISA write and read commands:



The final approach may be best suited for integration with an existing LabVIEW driver for controlling the X-App measurements.

Note that in all of the above examples, the hardware needs to be unlocked before the X-App can use it and then is locked back to the driver after the X-App measurements are complete.

The example is completed by allowing manual control of the X-App and then finally closing the VSG, VSA and X-Apps, clearing the errors, as discussed above:

