Getting Started with the Audio Precision APx LabVIEW .NET Driver

version 4.5, September 2017

Introduction

The APx LabVIEW .NET Driver is a collection of Virtual Instruments (VIs) that provides the ability to access most of the functionality available in the APx500 Application Programming Interface (API) using high level, LabVIEW-typical subVIs, with a minimum number of programming steps. Using the driver collection, you can quickly and efficiently develop APx applications with clean, concise LabVIEW code.

Contents

How to Use This Document

This document is a guide to getting started with the APx LabVIEW .NET Driver. In addition to providing background information and an overview of the driver, it contains a tutorial that will guide you through the process of creating a series of LabVIEW programs to control and interact with an APx500 Series audio analyzer. The tutorial begins with the simplest possible APx LabVIEW program and gradually increases in complexity, introducing and explaining features of the driver along the way.

The tutorial section assumes that you have a basic level of proficiency in LabVIEW. If you are new to LabVIEW, you should work through the *Getting Started with LabVIEW* manual that matches the version of LabVIEW that you are using. This document also assumes that you have a basic level of proficiency in using the APx500 measurement software.

Note that due to continual product improvement, figures in this document taken from earlier versions of the software and driver may not exactly match the version that you are using.

Background

The APx500 Series of audio analyzers are controlled via the APx500 measurement software. The APx500 software has many test automation features built in, including customizable user prompts and the ability to call external applications. For those who want to go beyond the automation features built into the APx500 measurement software, a full-featured Application Programming Interface (API) is available. The API is built on the Microsoft .NET platform, allowing custom APx programs to be developed in any .NET capable language.

National Instruments' LabVIEW is a graphical programming language that uses block diagrams instead of text-based code to create applications. LabVIEW is a popular development platform in test automation circles because it has an extensive library of instrument drivers and a broad test development feature set. Because LabVIEW supports .NET connectivity, it is one of the many

programming languages that can be used to develop custom APx500 applications using the APx .NET-based API. For more information on LabVIEW, visit http://www.ni.com/labview.

Audio Precision provides code examples on our website that demonstrate how to use VB.NET and C#, to interact with the APx API. Examples using LabVIEW are installed with the APx LabVIEW .NET Driver and are available from the Examples sub-palette.

Every feature available in the APx API can be accessed from LabVIEW, using basic low level calls to the .NET properties and methods available in the API. However, due to the graphical nature of LabVIEW and the way in which it interfaces with .NET objects, APx LabVIEW programs developed using only these low level .NET calls would be considered "awkward" by many programmers accustomed to using LabVIEW.

The APx LabVIEW .NET Driver is a collection of Virtual Instruments (VIs) intended to eliminate this awkwardness, and to enhance the development of APx programs using LabVIEW. It provides LabVIEW users the ability to access most of the functionality available in the API using higher level LabVIEW-typical subVIs, with fewer program steps. As a result, APx LabVIEW applications can be developed much more quickly and efficiently, with cleaner, more concise LabVIEW code.

LabVIEW Version Requirements

The APx LabVIEW .NET Driver was developed in LabVIEW version 2012 for users having LabVIEW 2012 or later.

Driver Version Information

The APx LabVIEW .NET Driver is version-specific to the APx500 measurement software. For example, the APx LabVIEW .NET Driver 4.5 must be used with APx500 4.5. The revision number of the driver (4.5.x) is independent of APx500 and indicates a driver update.

What's New in Version 4.5?

Features added to the APx LabVIEW .NET Driver in version 4.5 include:

- Updated Bluetooth Configuration and Action/Status VIs to support the new Bluetooth Duo module.
- Added new VIs to configure the DUT Delay measurement settings in Sequence mode.
- Miscellaneous improvements and fixes.

Upgrading Existing LabVIEW Projects and VIs

Changes from 4.4. to 4.5

- The Bluetooth Configuration Parameters typedef cluster control used in the VIs that get and set Bluetooth configuration settings has been extensively modified to support the new Bluetooth features added in APx 4.5.

Changes from 4.3 to 4.4:

- The following VIs are no longer polymorphic and the same VI will now work for either Sequence and Bench mode. If you are using the Sequence or Bench-specific versions, which have been removed, LabVIEW will prompt for a replacement. For example, when LabVIEW cannot find APx500 Sequence – Auto Set Generator Level or APx500 Bench – Auto Set Generator Level when loading a project, browse and select APx500 – Auto Set Generator Level to have LabVIEW automatically fix the error.

  ⇒ APx500 Auto Set Generator Level

  ⇒ APx500 Utility-Get IO Connector Details

  ⇒ APx500 Utility-Get Output Connector Details

  ⇒ APx500 Utility-Get Input Connector Details

  ⇒ APx500 Config-SigPath GetSet Output Connector Type

  ⇒ APx500 Config-SigPath GetSet Input Connector Type

  ⇒ APx500 Config-SigPath GetSet Filters Analog

  ⇒ APx500 Config-SigPath GetSet Filters Digital

Changes from 4.2 to 4.3:

- The settings control Type Defs for the Measurement Recorder, Signal Acquisition, and Signal Analyzer measurements have changed slightly due to the addition of the Loop Waveform checkbox. If you are using a control or constant that is not linked to the corresponding Type Def, you will need to update it manually.

- Some VIs had input and output terminals with the same name. These terminal names are now appended with "in" and "out". When using these VIs in TestStand, it may be necessary to refresh each step to update the names.

Changes from 4.1 to 4.2:

1. **New install location:**

   The APx LabVIEW Driver is now installed at `instr.lib\Audio Precision APx .NET` instead of at `instr.lib\Audio Precision\APx Driver`. In order to resolve conflicts due to the new location, you will need to do the following when opening a VI saved for an earlier version of the APx .NET Driver:

   a. LabVIEW will search and find the driver VIs in the new locations and display a prompt for you to accept each one. Accept the changes until the VI finishes opening.

   b. Save the project and all VIs, and then go to Project > Resolve Conflicts to resolve any remaining location conflicts.

   c. To ensure that no conflicts are remaining in memory, save the project and all its files again, and then close and reopen LabVIEW.

d. Before reopening the project, it is a good idea to do a Mass Compile on the entire project directory (Tools > Advanced > Mass Compile) to make sure that references in all VIs, including those that don't currently have callers, are properly updated.

2. Result types are now static objects (the MeasurementResultType enum is no longer used).

The .NET invoke nodes APx500.AddResult and APx500.DeleteResult will be broken and will show in the VI error list. To resolve the errors, do the following:

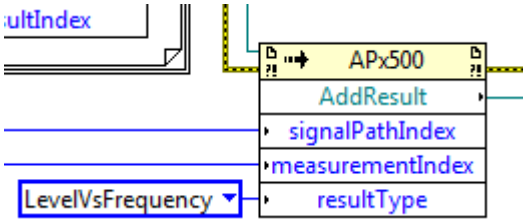


Figure 1. The AddResult invoke node from a previous project is broken in 4.2.

a. Right click on the node and choose "Select Method". Then reselect the desired method (AddResult or DeleteResult). This will fix the node but break the wire from the enum.

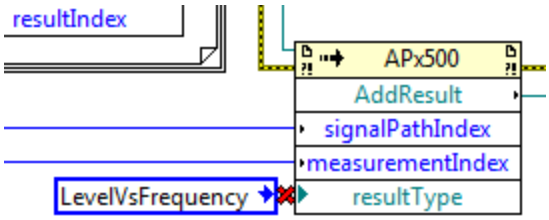


Figure 2. Reselecting the method fixes the invoke node but then breaks the wire from the MeasurementResultType enum.

b. Note the enum value (in this case "LevelVsFrequency"). Delete the enum and the broken wire.

c. Right-click on the terminal of resultType, select "Create property for …", and then select the static object of the desired result type. In this case, it will be "(S) LevelVsFrequency". Then wire it to resultType. Note that because this is a static property, it can be used without creating a class object. Therefore, the incoming and outgoing object reference terminals will remain disconnected. This will not cause a LabVIEW error.

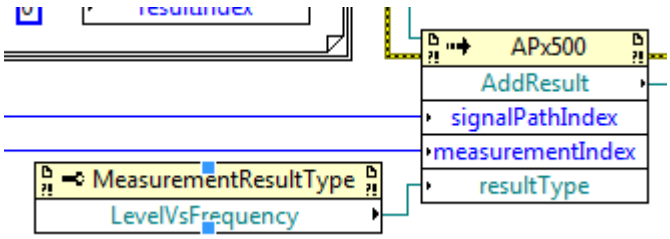


Figure 3. A MeasurementResultType static object is created to replace the enum.

d. Rerun the error cluster wire to include the new static object.

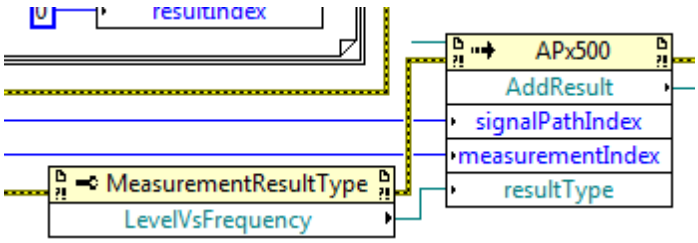


Figure 4. The error cluster is reconnected to include the static object.

Changes from 3.4 to 4.1:

If your project was saved using APx Driver 3.4.3 or earlier, you will need to note the following changes as well as those described in the preceding section.

- Controls and indicators have been reorganized into more logical front panel clusters.
- The Level result has been renamed RMS Level.
- *Reference Levels* is no longer a separate "measurement" but is incorporated into *Signal Path Setup*.
- Input bandwidth and filter controls have changed in APx500 v4.0 and v4.1. In the driver, these controls are no longer in the Input Configuration VIs and have been moved to separate Analog and Digital filter VIs.
- There are some changes to the underlying APx500 API. The most significant is that Generator.AnalogLevels and Generator.DigitalLevels is now just Generator.Levels. If you have any custom VIs you've created that use these properties they will need to be updated. For more details on API changes, refer to the APx API Browser.
- Because the driver now supports multiple channels for generator level, when writing a single value you will need to convert it to an array. The figure below shows how to create an array for setting the Frequency Response measurement analog generator level:

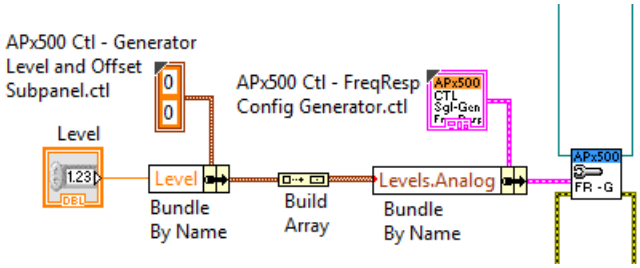


Figure 5. Writing a single value for analog generator level to the Frequency Response generator settings VI.

- Because the driver now supports multiple channels for generator level, when reading a single value you will need to convert it from an array. The figure below shows how to extract the first value from an array when reading the Frequency Response measurement analog generator level:

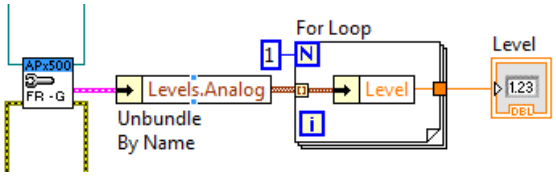


Figure 6. Reading a single value (channel 1) for analog generator level from the Frequency Response generator settings VI.

- When a compiled LabVIEW application is run on a Windows 8 or 8.1 PC, it may not be able to locate the APx API DLL. To resolve this issue, copy the APx API ({Program Files}\Audio Precision\APx500 4.5\API\AudioPrecision.API.dll) into the same directory as the application's executable.

Features

The APx LabVIEW .NET Driver was designed to follow the National Instruments LabVIEW Plug & Play Instrument Driver standard. Driver features include:

- Works with all audio analyzers in the APx500 Series.
- Supports configuration, running, and data handling of virtually all measurements in the APx500 measurement collection.
- Measurements can either be run individually or as part of a sequence defined in a project file.
- The ability to control any non-advanced[1] measurement setting.

Limitations

There are some limitations to the driver, including:

- Generally, advanced measurement configuration settings are not supported in the driver[1].
- Digital Serial Input and Output configuration is not supported. However, opening preset configuration files is.

Installation

If you have an earlier version of the APx LabVIEW .NET Driver installed, it is recommended that you uninstall it before installing this version. The driver can be uninstalled using the Add or Remove Programs feature in the Windows Control Panel, as shown in Figure 7.

[1] Note: Access to more advanced API features that are not implemented in the driver is still available using basic low-level .NET API calls. See page 57.

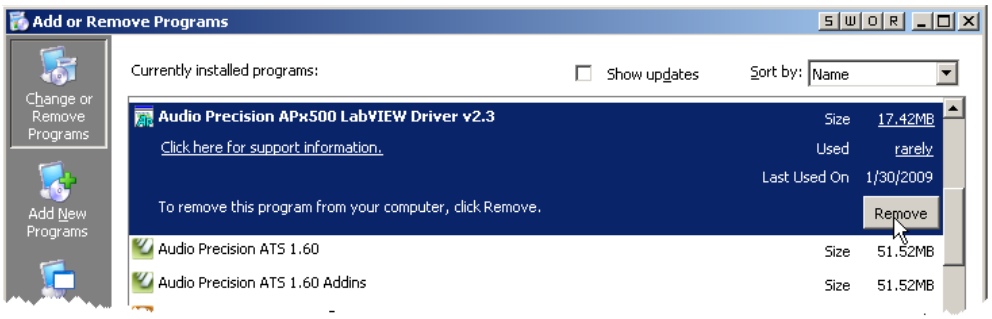


Figure 7. Uninstalling the previous version.

To install the APx LabVIEW .NET Driver, download it from ap.com, unzip it, and run setup.exe. By default, the installer will install the driver in the instr.lib sub folder of the LabVIEW installation on the computer, as shown below. This default location should not be changed. Otherwise, the driver's menu palette will not be available in LabVIEW.

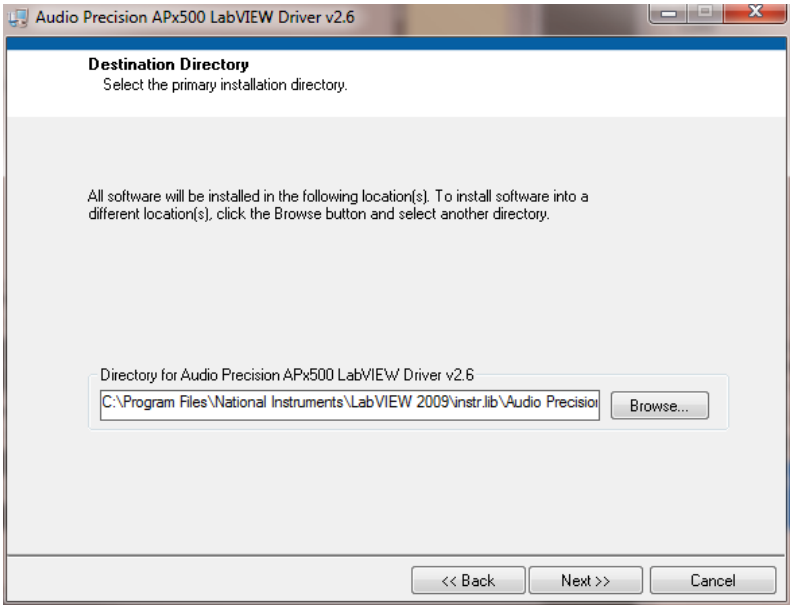


Figure 8. The APx LabVIEW .NET Driver installer.

Running LabVIEW in Administrator Mode

Normally it should not be necessary to run LabVIEW as an administrator when using this driver. However, if UAC (User Account Control) is enabled in Windows, in some cases it may be necessary to run as an administrator when calling an application that needs elevated permissions. One such example is the WaveReader VIs (see APx500 Example – WaveReader.vi), where it is necessary to run LabVIEW as administrator when Measurement Recorder is saving a wave file to the user documents directory.

Organization of the Driver

After installation, the driver collection will be contained in a folder named Audio Precision\APx Driver within the instr.lib sub-folder of the directory where LabVIEW is installed. For LabVIEW version 2012, the folder name is C:\Program Files\National Instruments\LabVIEW 2012\instr.lib\Audio Precision APx .NET.

The driver includes a special menu palette, accessible from the LabVIEW functions palette on the block diagram, under Instrument I/O - Instrument Driver - APx500, as shown in Figure 9.

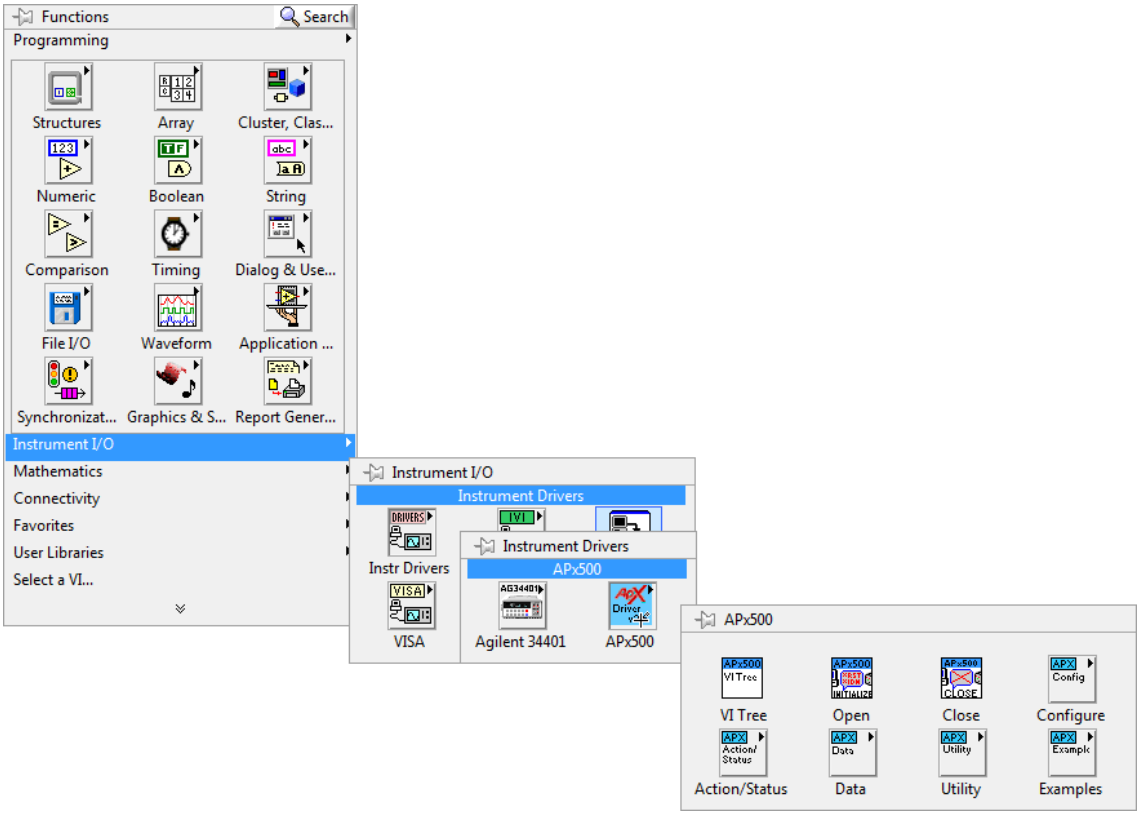


Figure 9. The APx LabVIEW .NET Driver menu palette

On the main menu palette, there is a link to a VI named APx500 VI Tree.vi. This is a special type of VI provided with LabVIEW drivers to help document the VIs in the driver collection. When opened, the front panel of this VI appears as shown in Figure 10. The VI Tree is not meant to be executable, as evidenced by the broken run arrow on the toolbar below the View menu item. The block diagram for this VI shows every high level VI in the driver (Figure 11).

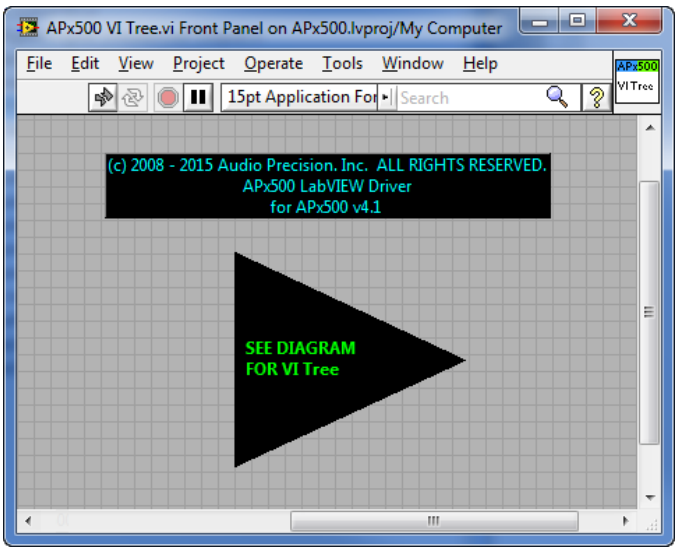


Figure 10. The VI Tree front panel

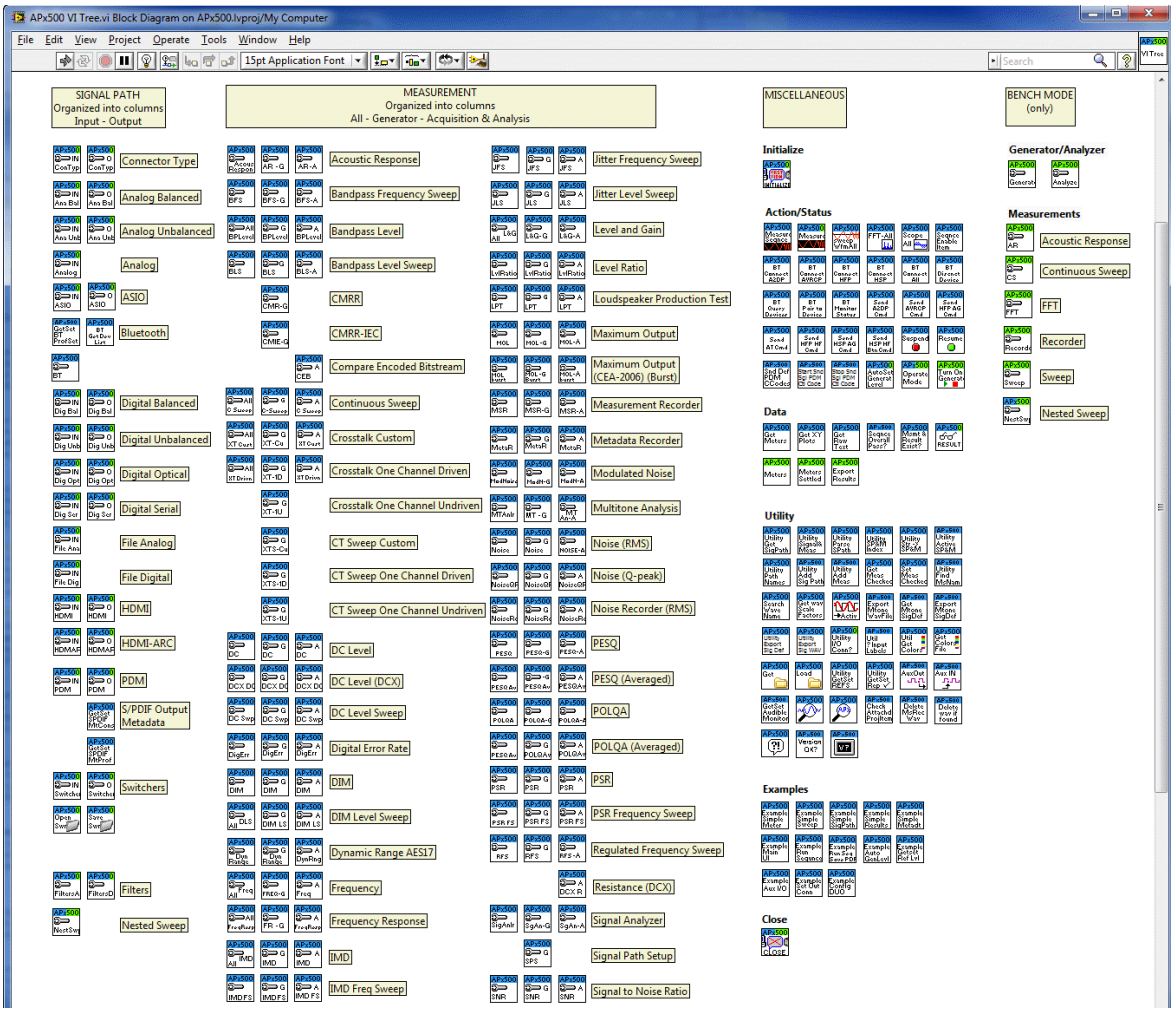


Figure 11. The VI Tree block diagram

Consistent with the National Instruments guidelines for an instrument driver, the VIs in the in the APx500 VI Tree (Figure 11) are organized into categories including Initialize, Configure, Action/Status, Data, Examples, Close, and Bench. These same categories are visible in the driver's menu palette (Figure 9).

Driver VIs may be loaded from the driver's menu palette or copied from the VI Tree.

The addition of Bench mode introduces a new icon color coding scheme as follows:

Table 1. VI icon color code

Color band on VI icon		APx500 mode:
Blue		Sequence mode
Blue with Green in corner		Polymorphic, with Sequence mode as the default. To switch to Bench mode, right click the VI and choose "Select Type > Bench." When switched to Bench mode, the color will change to Green.
Half Blue, half Green		Work as-is in either Sequence or Bench mode.
Green		Bench mode

Use a LabVIEW Project and Corresponding .config File

We have found that when working with .NET assemblies such as the APx500 API, the use of a LabVIEW project is critical.

The LabVIEW Driver installer puts a LabVIEW project named "*APx500 Examples Project.lvproj*" in the driver folder. There is also a shortcut to this project placed in the Start Menu. The Examples project contains a number of example VIs to help you get started using the APx driver.

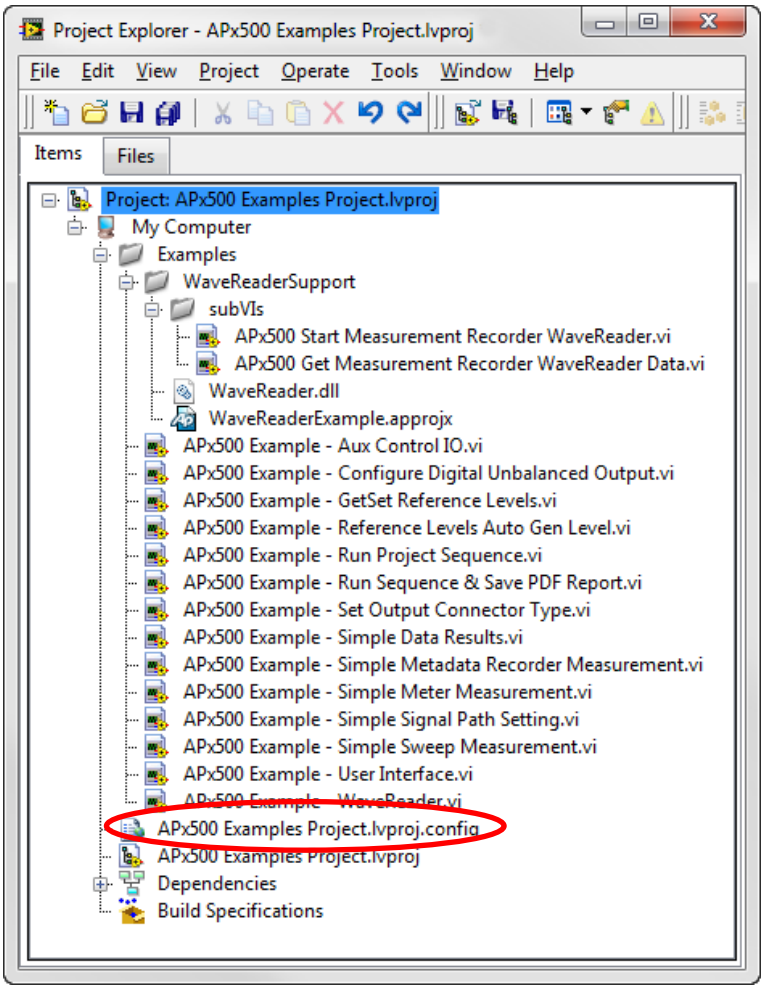


Figure 12

A configuration file with the same name as your LabVIEW project must be included in the folder that the LabVIEW project resides in. Using a project file and accompanying .config file like this when working with the APx Driver VIs will prevent errors caused by cross-linking and by LabVIEW referencing the wrong version of the APx API. If a project file and .config file are not used, in some cases the LabVIEW driver VIs may not work.

The Examples project shown in Figure 12 contains a configuration file named "*APx500 Examples Project.lvproj.config*". You can use this file in your own LabVIEW project by simply renaming it. The config file must have the same name as your LabVIEW project, followed by "*.config*". For example, if you create a new LabVIEW project named *MyAPx500Project.lvproj*, make a copy of the .config file above in the same folder as the project and rename it to *MyAPx500Project.lvproj.config*.

When updating a project for a new version of APx500, be sure to change the version references in the .config file accordingly. The example below shows the proper reference to the APx API version 4.5:

```
<bindingRedirect oldVersion="2.1.0.0-4.4.0.0" newVersion="4.5.0.0" />
```

The APx500 Open VI

The first VI in the collection is the APx500 Open.vi (Figure 13). This VI Opens the APx500 measurement software (if it is not already open), and creates a .NET reference to the APx API. This VI must be run before any of the other VIs in the driver collection, so that it can pass a reference to the API to any subVIs further down the line. Aside from the Example VIs, this is the only VI in the driver collection that can be run on its own as a Top Level VI (i.e., not as a subVI).

Note: If the APx500 application is already open, the APx500 Open VI simply creates a .NET reference to the API. It does not open multiple instances of the APx500 application.

The APx500 Open VI is polymorphic, allowing you to open APx500 in either Sequence or Bench mode. The default is Sequence mode. To open in Bench mode, add the VI to your block diagram, right click on it, and choose "Select Type > Bench."

LabVIEW's context help can be turned on or off by selecting Show Context Help from the Help menu, or by pressing Ctrl-H. Figure 13 shows the wiring diagram part of the context help for the APx500 Open.vi.

Figure 13

By default, the APx500 Open VI makes the APx500 application visible. If you prefer to use the API with the APx500 Application invisible, you can simply wire a Boolean False to the Visible input of the APx500 Open VI.

When the APx500 Open VI is added to a diagram, it will refer to the version of the APx LabVIEW .NET Driver that is currently installed. When you upgrade APx500 and open an existing project, LabVIEW will prompt you that the APx500 API version has been changed. You can check the API version by locating the APx500 constructor in the block diagram of the APx Open VI.

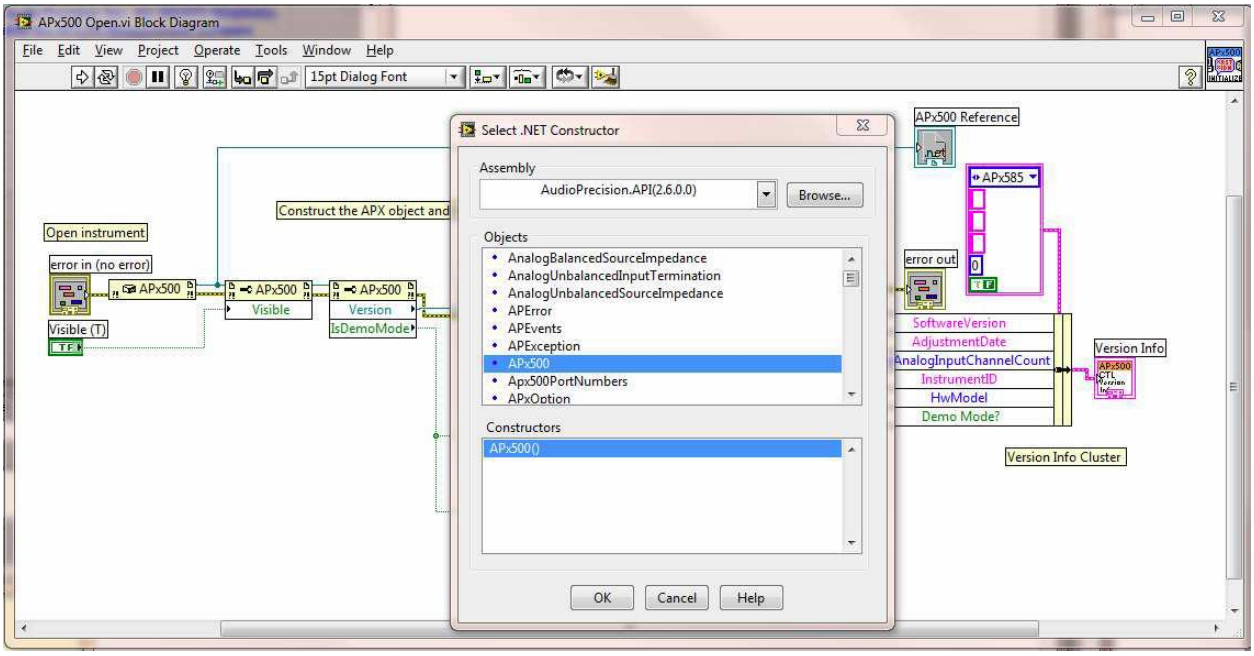


Figure 14

Getting Started–A First Simple APx LabVIEW VI

To create a simple APx LabVIEW program, open a New VI, and place a copy of the APx500 Open.vi on the block diagram. Next, right-click on the Version Info output tab of the VI and select Create – Indicator, to create a Version Info indicator. Finally, for good measure, add a Simple Error Handler VI to the diagram and connect it to the error out indicator of the APx500 Open VI. When you are finished, your block diagram should be similar to Figure 15.

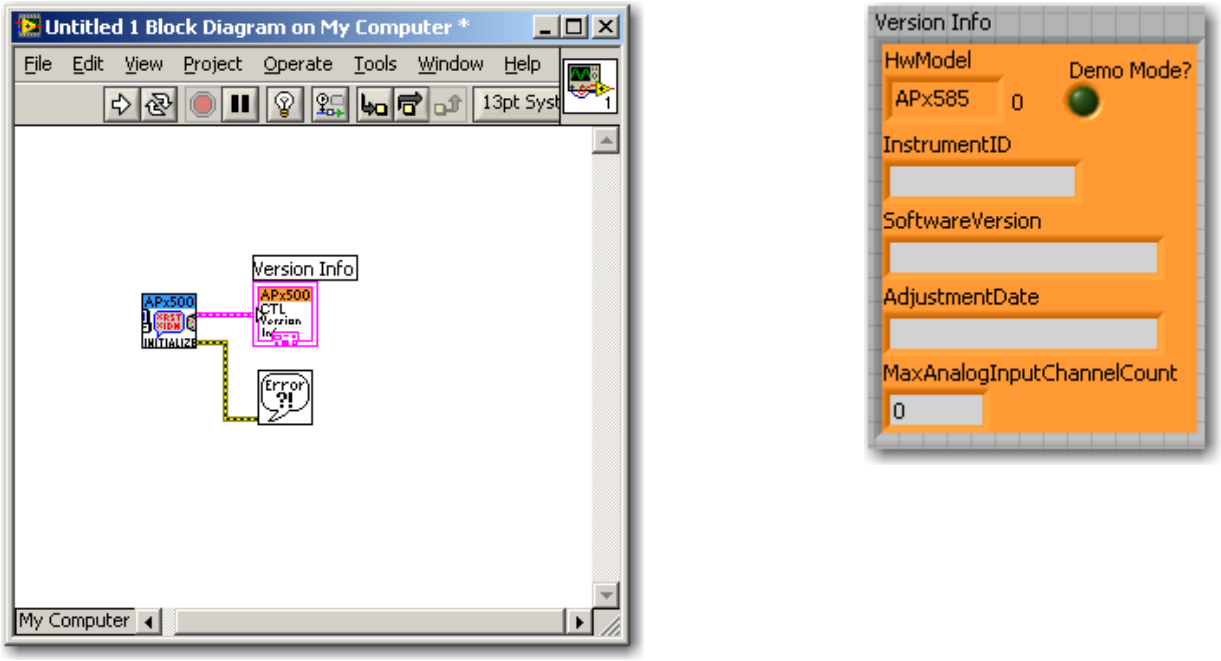


Figure 15. Block diagram of a first simple VI to control an APx analyzer, and the Version Info cluster on the front panel

This simple VI is now ready to run. Note that if the PC is not properly connected to an APx500 analyzer that is powered on, the APx500 measurement software can be run in demo mode. In fact, if the APx application is loaded from the API (e.g., using LabVIEW) rather than from Windows, the software is automatically loaded in demo mode when an instrument is not connected. Demo mode simulates almost all of the functionality of the APx software, but returns random data. Fortunately, all of the API calls function normally in demo mode, and therefore, LabVIEW VIs to control APx analyzers can be mostly developed without an APx analyzer being connected.

To run this simple VI, click the white Run arrow on the VI's front panel. After several seconds (the amount of time depends on the PC speed and available memory resources), the APx application will load, and the Version Info will be displayed in the indicator on the front panel[2]. As explained in the context help, if the APx application is running in demo mode, some of the indicators in the Version Info cluster return blank values.

Using the APx500 Close VI

The APx500 Close.vi closes the APx500 measurement software and closes the .NET reference to the APx API. With traditional LabVIEW instrument drivers, it is considered good practice to close any driver references when a VI is finished its task. Figure 16 shows the block diagram of the simple VI from Figure 15 with the Close VI added.

[2] Note: An exception to this occurs the first time an APx analyzer is used with a new version of the APx500 application. In this situation, the application pauses to update the instrument's firmware.

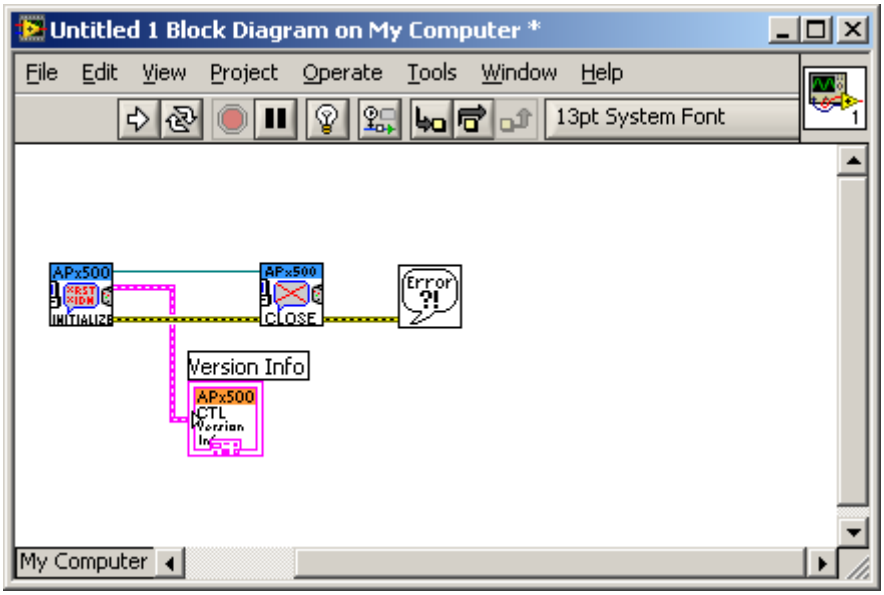


Figure 16. APx500 Close VI added to the she simple VI.

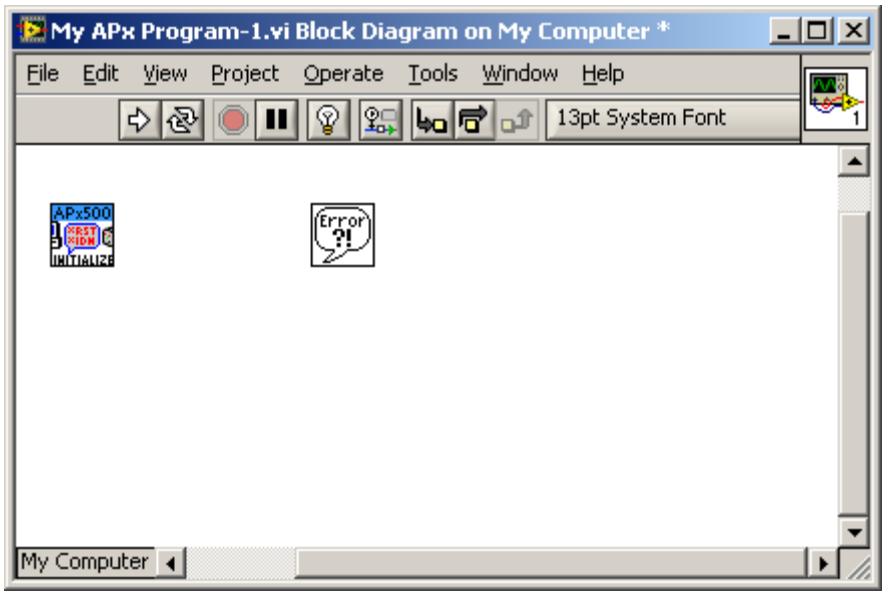
The APx LabVIEW .NET Driver is not a traditional LabVIEW instrument driver in the sense that running the APx500 Open VI without running the Close VI does not open an additional instance of the APx500 measurement software. Therefore, there is no advantage to adding the APx500 Close VI to every VI. In fact, closing the application every time a VI finishes running can be a disadvantage, because of the time it takes to re-open the application. We do not recommend adding the APx500 Close VI to every top level VI.

The only potential disadvantage to the above strategy is if the APx500 application is running with its UI invisible when you are finished using it with LabVIEW. In this case, if the APx500 Close VI is not used somewhere in the LabVIEW program, the APx500 application will remain in memory without the user knowing it, because the UI window is invisible. If you are concerned about this you should add the APx500 Close VI to your APx VIs. However, in our opinion, the inconvenience of having to re-open the APx500 application every time a VI is run far outweighs the disadvantage of the application occasionally being left running and invisible.

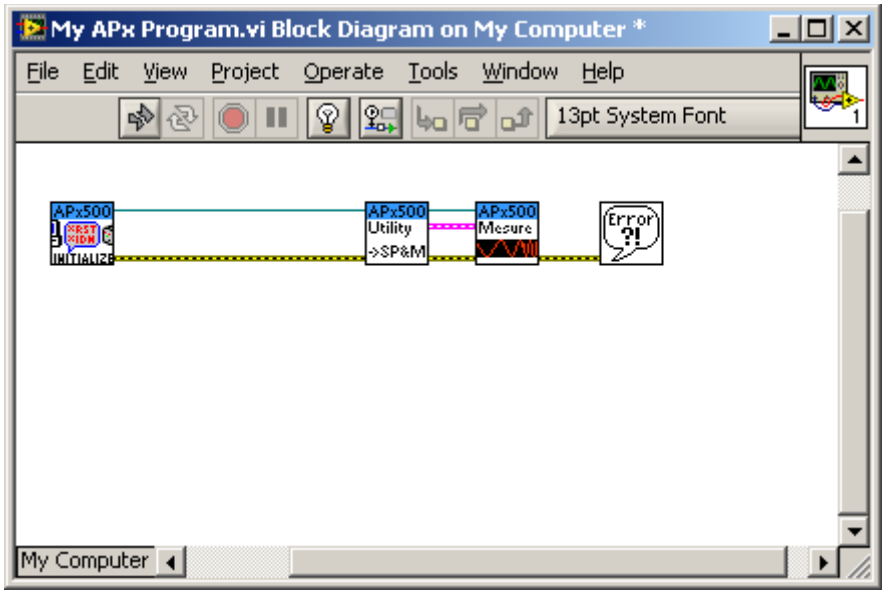
Running a Measurement in the APx Project

Next, we will look at running an APx measurement from LabVIEW. Begin by saving the simple VI created above. In the examples shown below, the VI has been saved with the name My APx Program.vi. Follow the steps below to create a simple LabVIEW program to run a Level and Gain measurement on the APx analyzer.

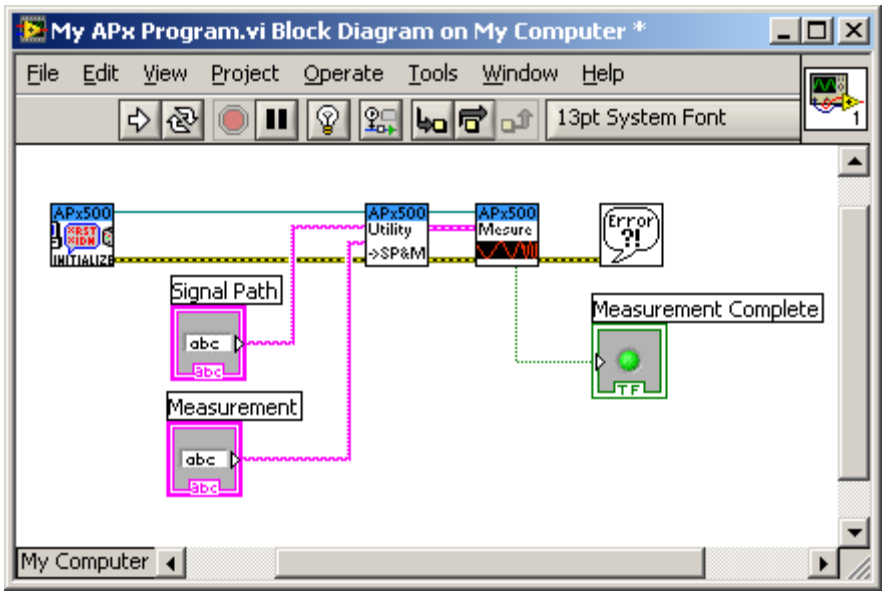
1. Delete all sub-VIs from the block diagram, except the APx500 Open VI and the Simple Error Handler



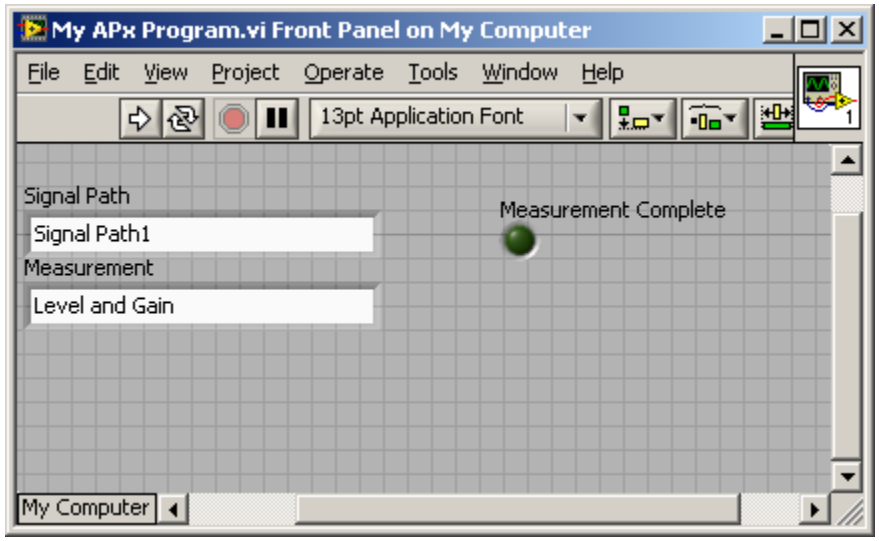
2. Add the *APx500 Utility – Strings to Signal Path & Measurement* VI and the *APx500 Perform Measurement* VI to the diagram and connect them as shown.



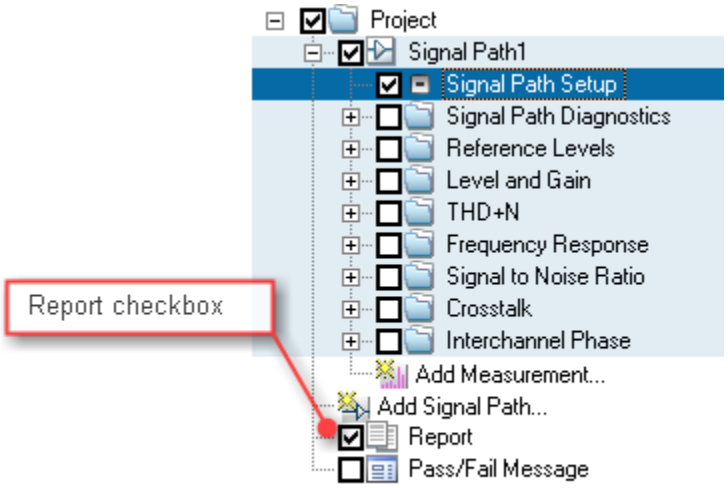
3. Right-click on the appropriate inputs to the *Utility – Strings to Signal Path & Measurement* VI, and create controls for the Signal Path and Measurement name strings.
4. Right-click on the Measurement Complete output of the *Perform Measurement* VI and create an indicator.



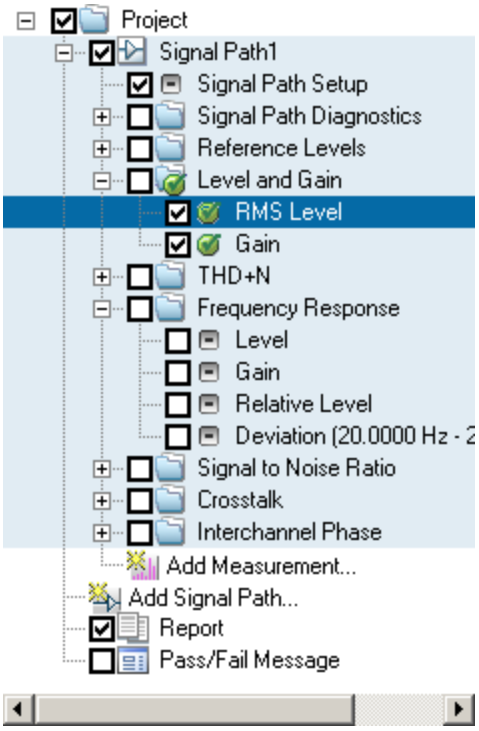
5. Enter the names "Signal Path1" and "Level and Gain" in the appropriate string controls on the front panel of the VI, as shown.



6. Open the APx500 application (if it is not already open). If it is open, start a new Project with the Default template. This will ensure that a Signal Path named “Signal Path1” exists in the Project and that a measurement named “Level and Gain” exists within that Signal Path. It will also ensure that the Report checkbox is checked.



7. Run the VI by clicking the Run arrow. This will cause the APx500 application to complete a Level and Gain measurement and create a report. Once complete, the RMS Level result of the Level and Gain measurement will be highlighted in the APx Application. On the front panel of the VI, the Measurement Complete LED should now be green instead of grey.



The above simple program illustrates running an APx measurement from LabVIEW. The Perform Measurement VI uses the APx Sequencer to run the measurement. This is equivalent to checking the measurement’s checkbox in the APx Navigator, and then right-clicking on the measurement and selecting Run Measurement (Figure 17). Running the measurement via the Sequencer is preferred over simply turning on the generator and reading the instantaneous data, because the Sequencer method returns settled readings.

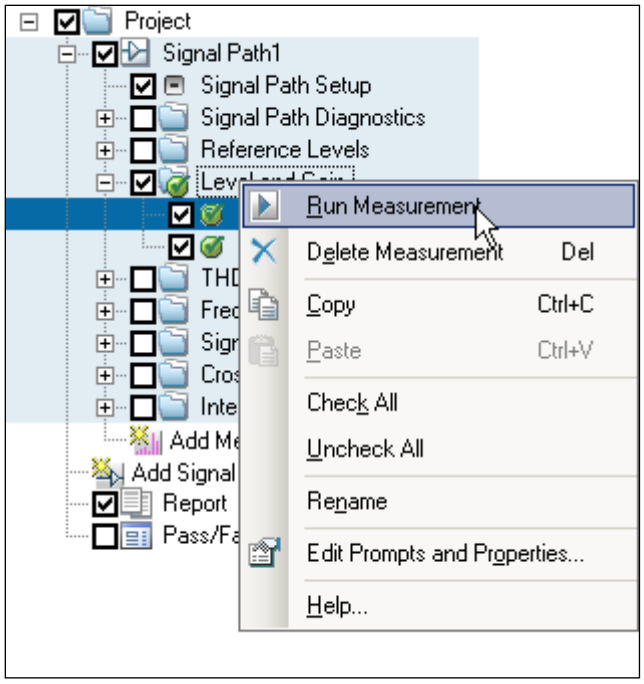


Figure 17

Note that specifying the Signal Path and measurement name is a critical step. This tells the APx API precisely which measurement in APx that our LabVIEW program wants to interact with. It is critical that these two items are specified correctly, and that they exist in the project. Otherwise, the API will generate an error. To see this in action, change the string in the Signal Path control on the LabVIEW front panel from "Signal Path1" to "Signal Path2", and run the VI again. This time it will generate the error message shown in Figure 18.

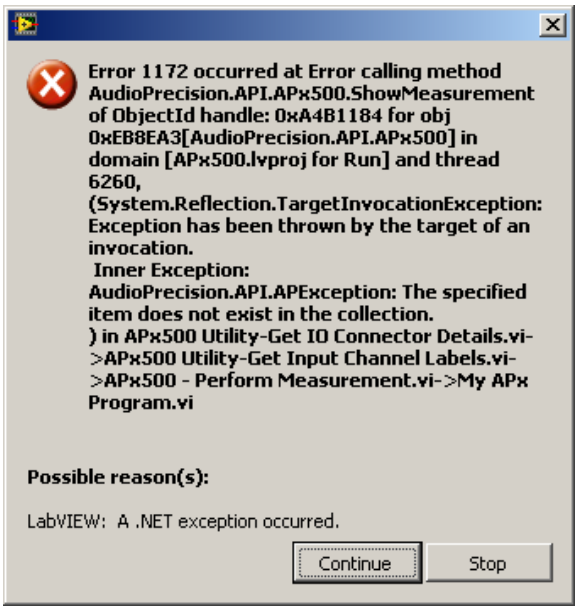


Figure 18

The same error message is generated if the measurement name specified in the Measurement string control does not match a measurement in the specified Signal Path. Therefore, you must ensure that the specified Signal Path name exists within the APx project, and that the specified measurement name exists within that Signal Path.

Error Handling

All of the error handling in the APx LabVIEW .NET Driver takes place in the APx API. The LabVIEW VIs in the driver collection merely pass the LabVIEW error cluster down the line. Every VI in the driver collection has an error in control and an error out indicator. To follow recommended programming best practice, you should wire the error in and error out terminals of all driver VIs, and add an error handler VI to your top level VI.

Unfortunately, LabVIEW does not do a very good job of handling .NET exceptions. As described in the LabVIEW Help system and on the NI Support site, any exception thrown when calling a .NET object or property is converted into LabVIEW Error 1172. LabVIEW does add more error information to the error message, but if you use LabVIEW's built-in Simple Error Handler VI, the error messages returned are rather cryptic and difficult to understand (see, for example, the error message in Figure 18). To overcome this problem, an interface called *APx.LastException* is part of the APx500 API. This function keeps track of the last .NET exception thrown by the APx500 application, giving LabVIEW access to the same managed error handling features available in .NET.

The VI named *APx500 Utility-APx Simple Error Handler* uses the *APx.LastException* interface to trap LabVIEW's generic 1172 errors and to return instead error messages that are much more meaningful and easy to read. The function of this VI is described in Figure 19. To compare error messages returned by this VI to those returned by LabVIEW, compare Figure 18 to Figure 20.

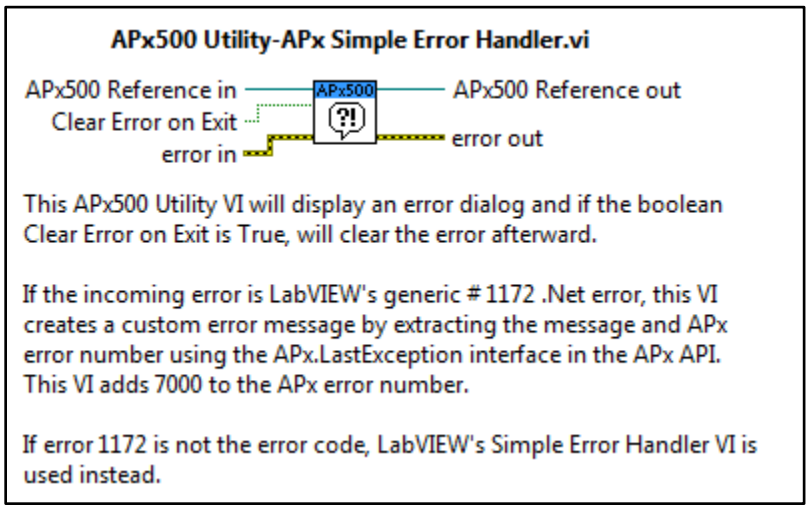


Figure 19

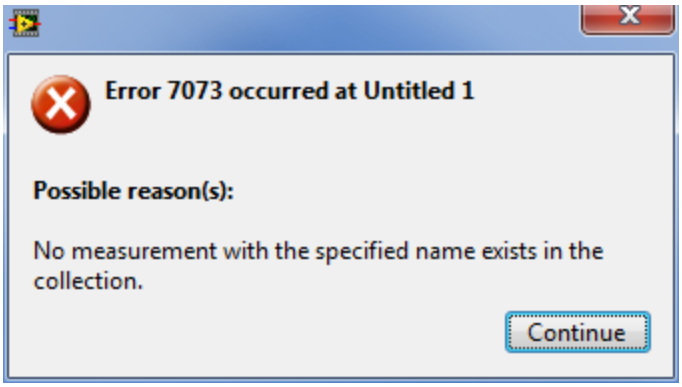


Figure 20

The Signal Path & Measurement Cluster

Due to the importance of specifying Signal Paths and measurement names correctly, the APx LabVIEW .NET Driver uses a special control named Signal Path & Measurement. In Figure 21, you can see that this control is passed into and out of the Perform Measurement VI. If you browse through the VIs in the driver collection, you will notice that Signal Path & Measurement is an input or output for many of the VIs that make up the driver.

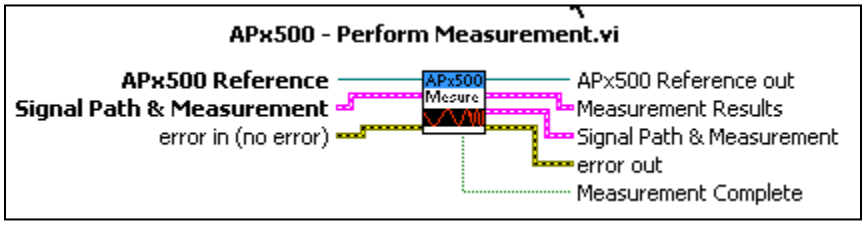


Figure 21

To take a closer look at this special control, add a Signal Path & Measurement indicator to the My APx Program.vi after the *Strings to Signal Path & Measurement* VI as shown in Figure 22 (right-click on the wire and select Create – Indicator). Next, change the Signal Path string back to "Signal Path1" and run the VI. The resulting front panel is shown in Figure 12 (controls have been rearranged for better visibility).

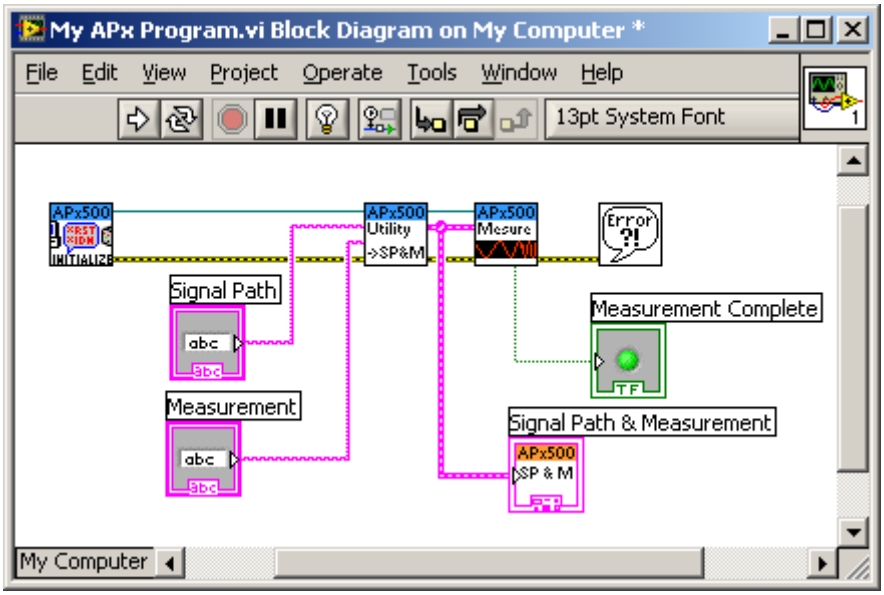


Figure 22

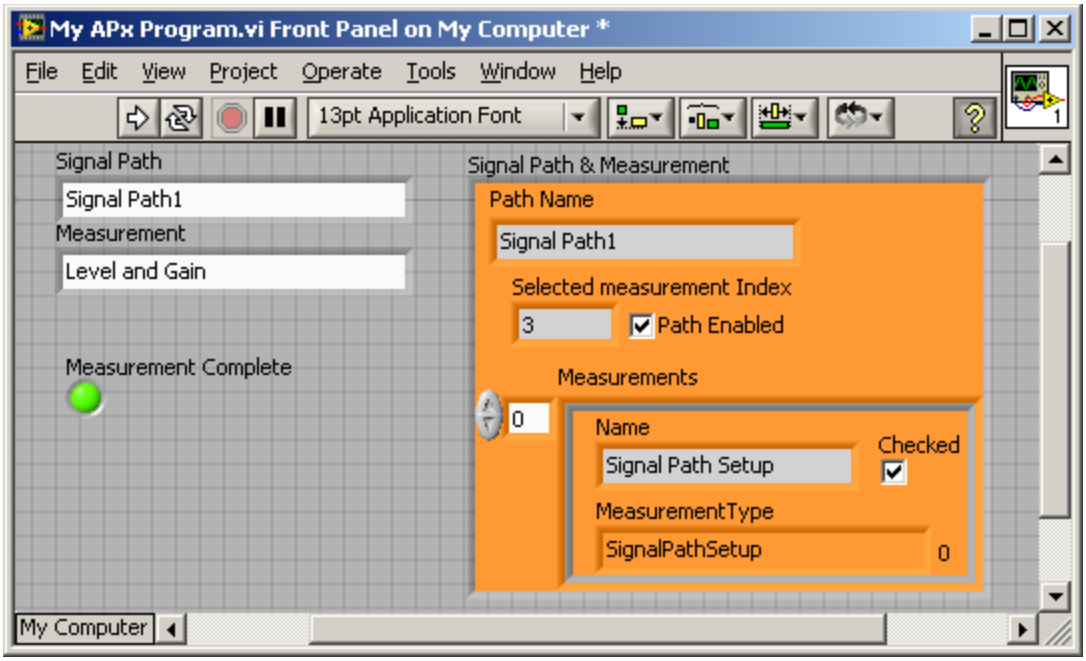


Figure 23. An indicator for the Signal Path & Measurement cluster added to the VI.

As shown in Figure 23, the Signal Path & Measurement indicator now contains an array of measurements. If you change the Measurements array index, you will see that each element in the array corresponds to a measurement in Signal Path1 of the currently active APx project. The array also contains a check box denoting whether the measurement is checked, and a Measurement Type (enumerated type) indicator.

One important feature of the Signal Path & Measurement cluster is not immediately obvious: The Selected Measurement Index control (shown below the Path Name) in the cluster, is the control used to specify the measurement within the Signal Path. This index is zero-based. In the example shown in Figure 23, the Selected Measurement Index is equal to 3. Note that this

corresponds to the position of the measurement in the Signal Path, with index 0 being the first measurement (Signal Path Setup), index 1 = Signal Path Diagnostics, index 2 = Reference Levels, and index 3 = Level & Gain.

There are many utility VIs in the driver collection for working with Signal Paths and measurements. For example, the Get Signal Paths VI (Figure 24) returns an array of Signal Path & Measurement clusters with one element for every Signal Path in the currently loaded APx project. This structure provides a way of addressing every measurement in a project. This utility will be useful when you create more advanced VIs as will be seen later in the tutorial.

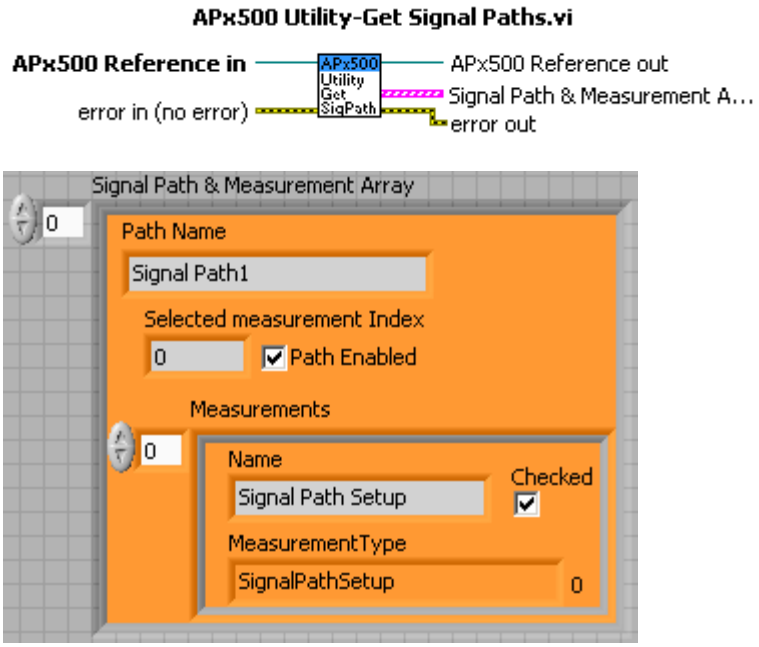


Figure 24

Changing APx Measurement Settings with the Driver VIs

The simple *My APx Program* VI above performs a measurement with the settings as configured in the APx project. But most users will want to configure the measurement from LabVIEW – for example, you may want to vary the generator level or frequency. So next we will look at using the driver VIs to change the measurement configuration.

In the *My APx Program* VI, delete the Signal Path & Measurement indicator. Next, add the VI named *APx500 Config-Level & Gain GetSet All* (available from the Configuration sub-palette, as shown in Figure 25) to the block diagram. Insert this VI between the *Utility – Strings to Signal Path & Measurement* VI and the *Perform Measurement* VI. Be sure to connect the input and output wires (the .NET reference, the Signal Path & Measurement and the error cluster). Next, add a Boolean True constant to the diagram and wire it to the Set connector on the VI. Finally, right-click on the Level & Gain Config input terminal of the VI just added, and select Create – Control, to create a Level & Gain Config control. The block diagram should now be similar to Figure 26.

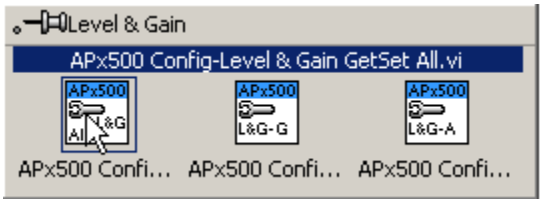


Figure 25

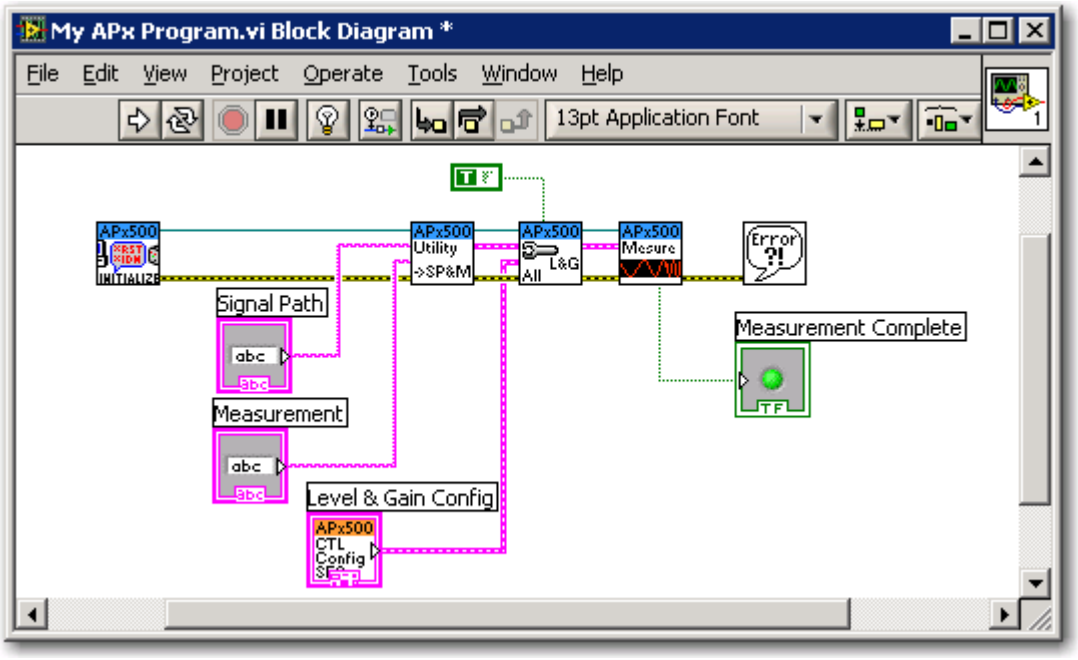


Figure 26

Now look at the Level & Gain Config control on the front panel of the VI. Figure 27 shows the front panel (you may need to rearrange controls and indicators to make your front panel match Figure 27). This control is a LabVIEW cluster control, that contains two other cluster controls - one at the top called Signal Generator Settings (L&G), and another below it called Signal AcqAnalysis Settings (L&G).

Now look at the APx UI. With the Level and Gain measurement highlighted in the Navigator, you may notice the similarity between the LabVIEW cluster controls and the APx controls used to configure the measurement, located in the grey panel to the right of the Navigator tree (Figure 28). The LabVIEW controls have been designed to have a one-to-one correspondence with the APx controls. This is obvious in the case of the Signal Acquisition and Analysis control, because there is only one control in this case (Low-pass Filter), and both LabVIEW and APx have the same control. But what about the Signal Generation Controls? Why are there more controls in the LabVIEW cluster than there are in APx? In fact, the APx UI has the same number of controls as the LabVIEW cluster, but they are not all visible at the same time. The APx500 application hides any controls that are not relevant in a given measurement context.

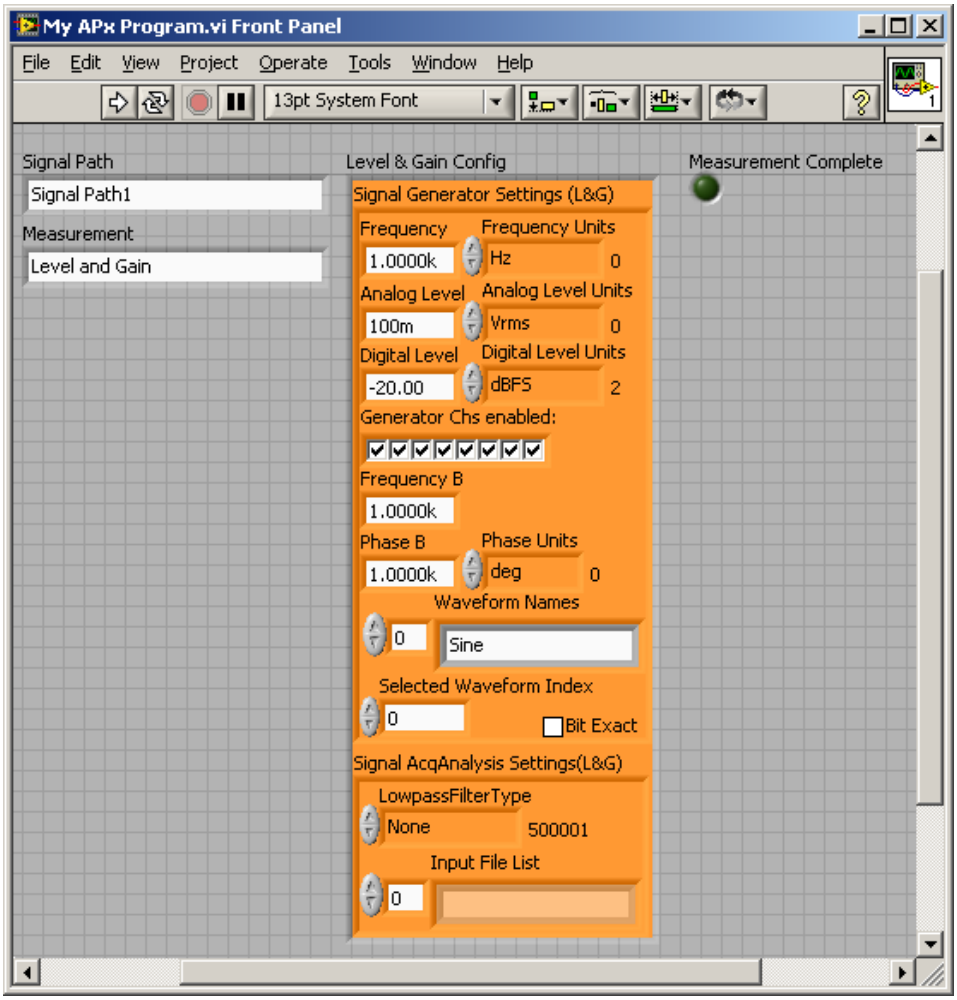

My APx Program.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt System Font
Signal Path
Signal Path1
Measurement
Level and Gain
Level & Gain Config
Signal Generator Settings (L&G)
Frequency
1.0000k
Frequency Units
Hz
0
Analog Level
100m
Analog Level Units
Vrms
0
Digital Level
-20.00
Digital Level Units
dBFS
2
Generator Chs enabled:
Frequency B
1.0000k
Phase B
1.0000k
Phase Units
deg
0
Waveform Names
0
Sine
Selected Waveform Index
0
Bit Exact
Signal AcqAnalysis Settings(L&G)
LowpassFilterType
None
500001
Input File List
0
Measurement Complete

Figure 27

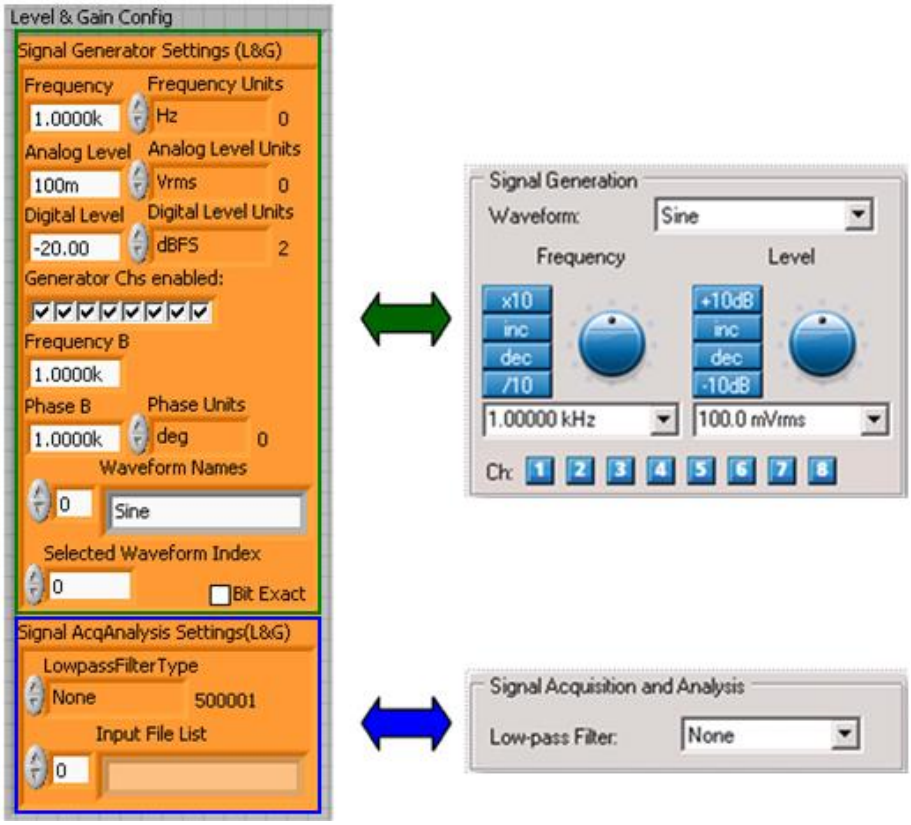
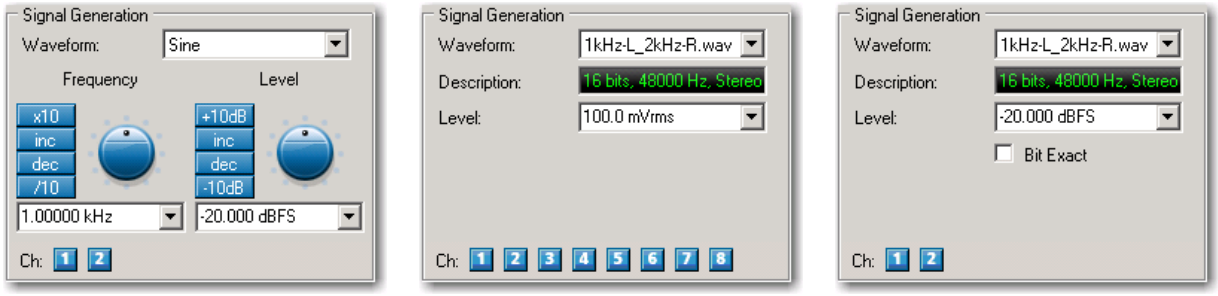


Figure 28

For example, Figure 28 shows the APx Signal Generation controls for the case of an Analog Output Connector type, when the Waveform type is Sine (the standard sine generator). Figure 29 shows other variations in the Signal Generation controls, depending on whether the Output type is analog or digital, and the Waveform type is Sine or Arbitrary Waveform.



a) Digital Output, Sine generator. b) Analog Output, Arbitrary waveform. c) Digital Output, Arbitrary Waveform

Figure 29. Variation in Signal Generation controls displayed by Output type and Waveform type

By studying Figure 28 and Figure 29, you can see that the LabVIEW Signal Generator Settings cluster contains all of the basic generator controls that could be needed for any combination of output type and waveform type. On first glance, it might appear that this doesn't make sense - for example, why make the digital level control visible when using the analog generator? However, it should be noted that (1) these are driver VIs that are not meant to be visible, not a User Interface, and (2) the controls must be present in order for LabVIEW to be able to pass them into

and out of a subVI. Hence, the controls for all signal path contexts are present, and it is up to you to ensure that your program uses the right controls.

How the Driver VIs Handle Units in Configuration Settings

The APx500 application has sophisticated controls that have the following features:

- Numerical value and units are combined in one control (e.g., 1.000 Vrms)
- Values are converted to SI formatting (e.g., 1.000 mVrms or 1.000 µVrms)
- Conversion among units occurs within the control (e.g., Vrms, Vp, Vp-p, dBV, etc.)

In the APx LabVIEW .NET Driver VIs, controls for setting the measurement configuration handle this by using a numerical control accompanied by a units control. Hence, as shown in Figure 30, to set the analog generator to 100 mVrms, you would set the Analog Level control to 100m, and the Analog Units control to Vrms. To set the analog generator level to -20 dBV, you would set the Analog Level control to -20.0 and the Analog Level Units control to dBV. Note that LabVIEW controls do support the use of SI formatting (100m = 0.100, 10.0k = 10,000, etc.)

Figure 30

Units are handled differently for results returned from APx. This will be discussed in the section on Accessing Measurement Results, later in this document.

Changing Measurement Settings–A Simple Example

Now that we've covered units, we are ready to try configuring the Level & Gain measurement settings using the *My APx Program* VI. When the Level & Gain Config cluster control was added to the VI, it had the default settings that a Level and Gain measurement would have when first added to the APx project. Open the APx500 application and ensure that the Connector in Output Configuration is set to Analog Unbalanced (Figure 31).

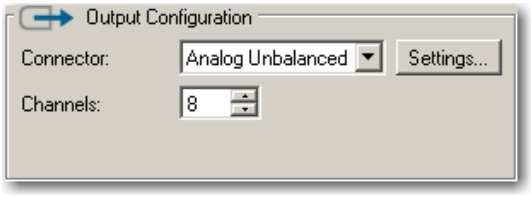


Figure 31

Next, change the values of some of the controls in the Level & Gain Config cluster on the front panel of the some of the *My APx Program* VI. For example, change the Analog Level to 200m, the Frequency to 2.0k, uncheck some of the checkboxes of the Generator Chs enabled control, and change the Low-pass Filter from None to 20 kHz (Figure 32). Now run the VI. The corresponding settings in the APx Level and Gain measurement will be changed and the measurement will be run.

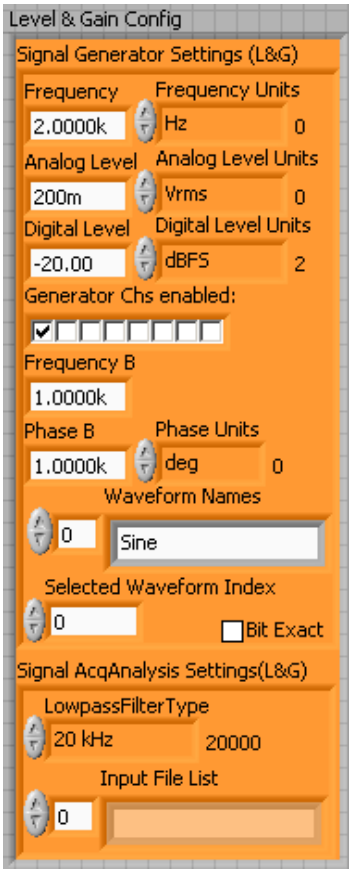


Figure 32

Changing Measurement Settings–A Better Example

The simple example above works, but it has the disadvantage that it will change ***all*** of the Level and Gain Generator and Signal Acquisition settings to the values in the LabVIEW cluster control. This is not how most users will want to interact with APx; most users will want to change only one or two settings – for example the generator level or the generator frequency. To accomplish this, the LabVIEW VI will have to read the current state from APx and then allow the user to change the one or two desired settings. The measurement configuration VIs were designed specifically with this in mind. To see how, let's take a look at the context help for the *APx500 Config-Level & Gain GetSet All* VI (Figure 33).

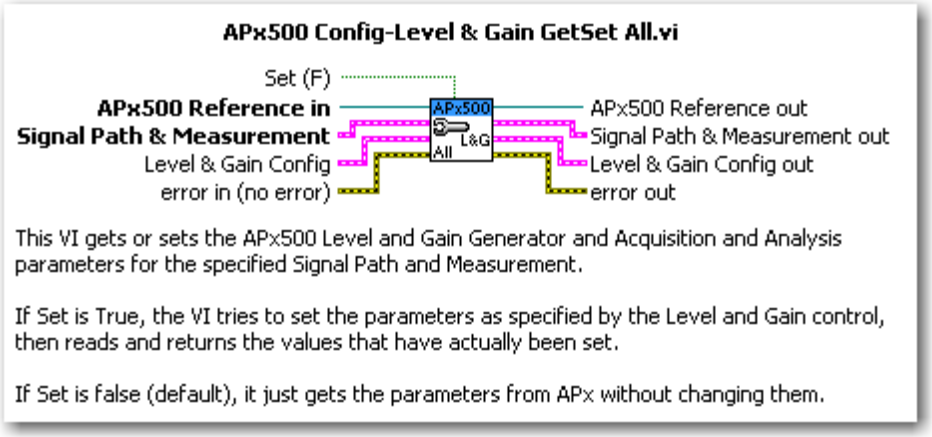


Figure 33

As shown, this VI has two functions: it can be used to *get* the configuration settings from APx or to *set* them. Note that the GetSet VI always reads the settings back from the APx500 application after setting them. This ensures that the state of the APx500 application is always maintained by the application itself rather than by LabVIEW. This helps to prevent confusion.

So let's look at how to use these Get and Set functions in a LabVIEW VI. We will modify the My APx Program VI to be interactive, to allow the user to change only the specific controls that they wish to. Proceed as follows.

1. In the *My APx Program* VI, delete the *Perform Measurement* VI, the Simple Error Handler VI and the Measurement Complete indicator. These will be added again later. Next, add white space to the diagram as shown in Figure 34 (e.g., using Ctrl-drag with the mouse).

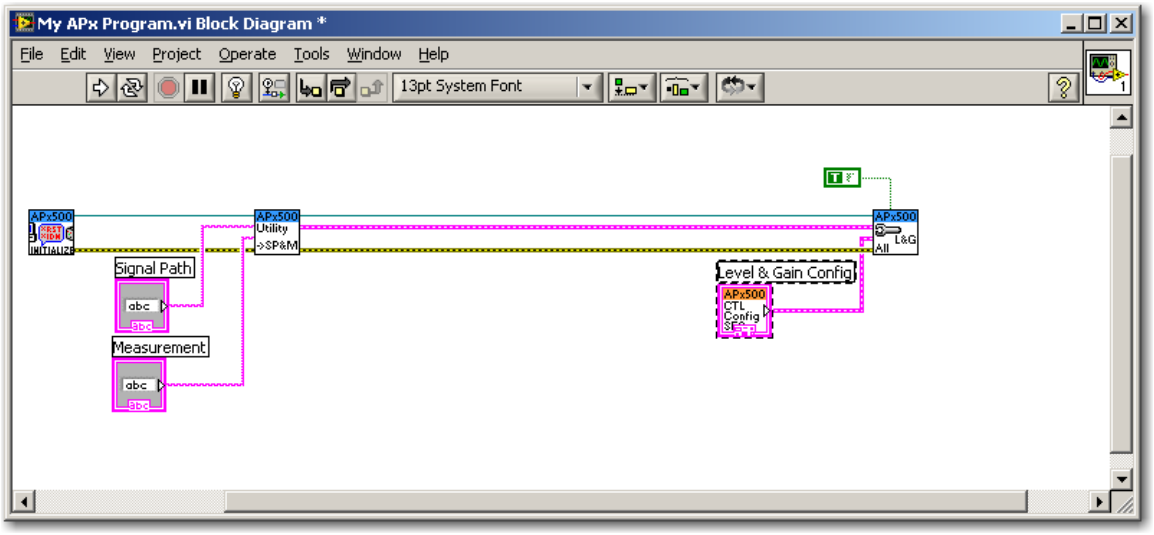


Figure 34

2. Add a Boolean button control to the front panel and label it "Update APx Settings" (Figure 35).

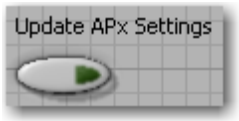


Figure 35

3. Add an Event Case around the *APx500 Config-Level & Gain GetSet All* VI, move the Update APx Settings control into that case, and change the event handled by the case to be a Value Change of the Update APx Settings control (Figure 36). When the modifications are finished, this will cause the configuration settings to be sent to APx when the button is pressed.

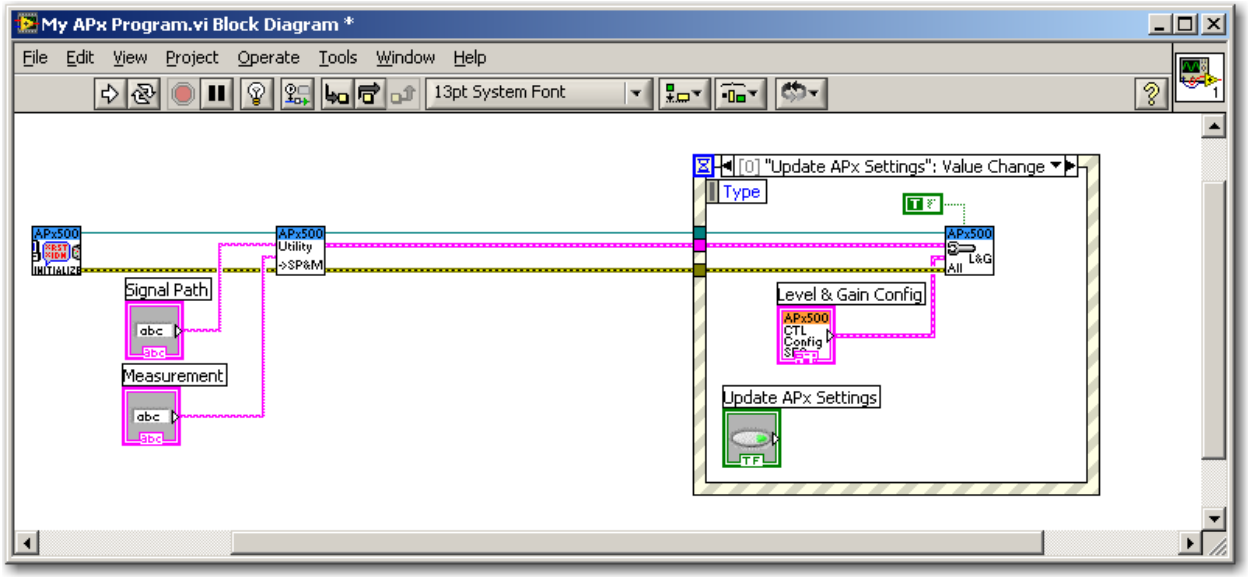


Figure 36

4. Add a While loop around the event case, change its input tunnels to shift registers, and then wire and rearrange the shift registers as shown in Figure 37.

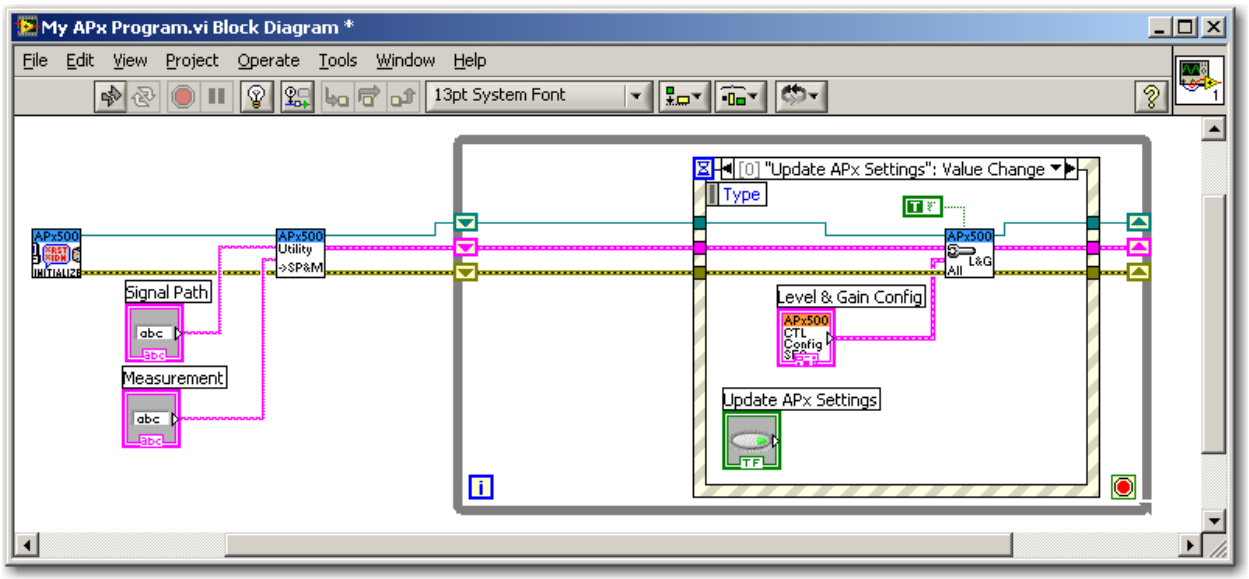


Figure 37

5. Create a local variable for the Level & Gain Config control as follows. Right-click on the control's terminal on the block diagram and select Create – Local Variable. Insert the local variable between the beginning of the While Loop and the Event Structure, and wire it as shown. You will need to add another shift register to the While Loop (Figure 38).

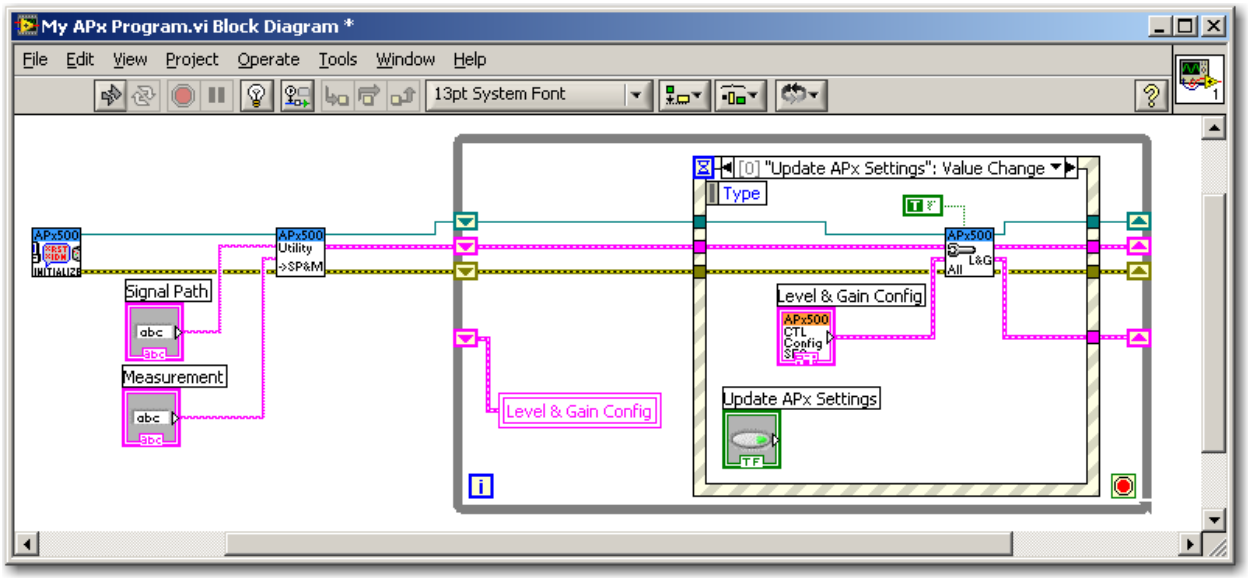


Figure 38

6. To make this VI function properly, we need to connect something to the input of the shift register created in step 5. Otherwise, the Level & Gain Config control will not be initialized properly. We want to initialize it to the values that are currently set in APx. Therefore, we need to *get* the values from APx. Add a copy of the *APx500 Config-Level & Gain GetSet All* VI to the diagram outside the While loop and wire it as shown in Figure 39. Note that the nothing is wired to the Boolean Set input at the top of the VI. Hence the default value (Get) will be used, causing the VI to get all the current settings from APx and initialize the Level & Gain Config control to those values.

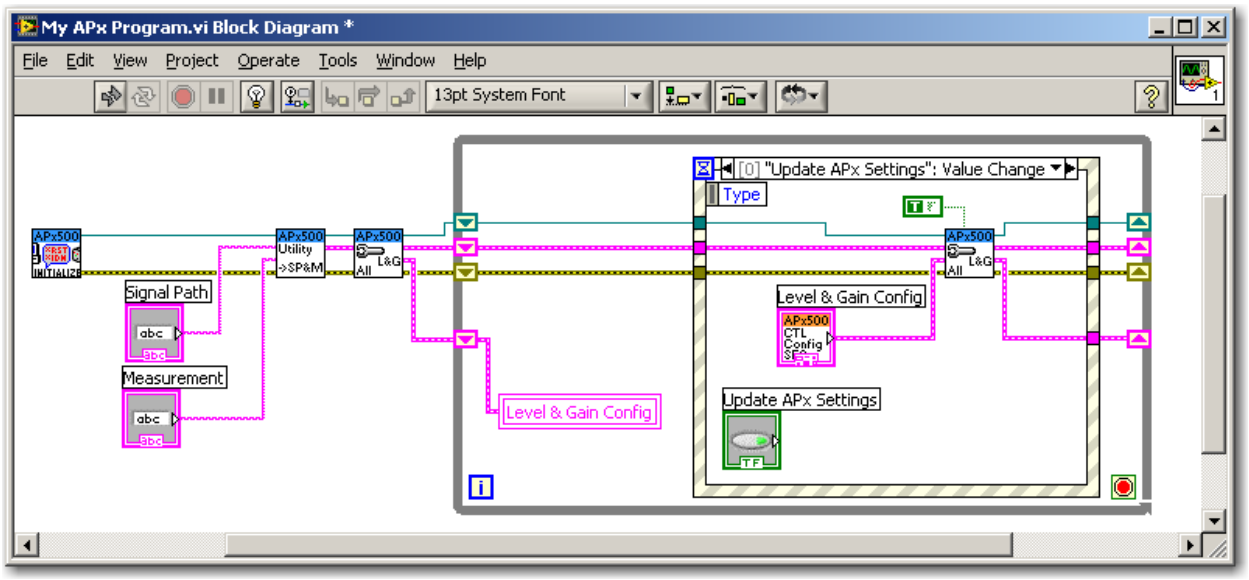


Figure 39

7. To make the VI functional, we need to wire something to the conditional terminal of the While loop. Create a stop button by right-clicking on the terminal and selecting Create – Control. Next, create a Stop case in the event structure, move the Stop control's terminal inside the case and wire it as shown in Figure 40.

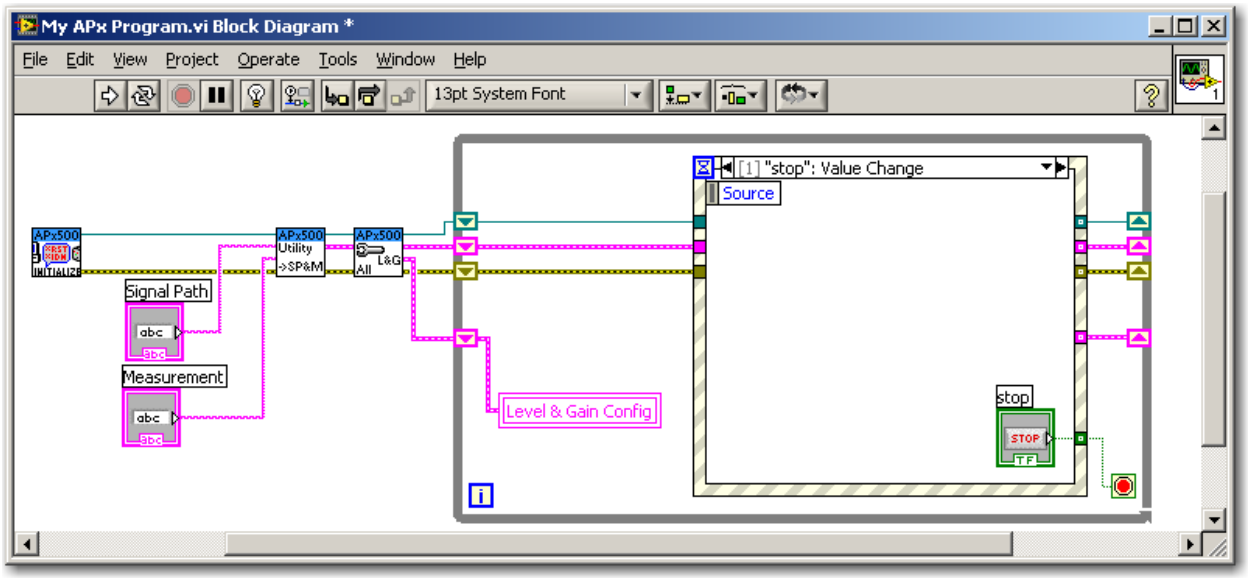


Figure 40

8. Next, we will add the capability to run the Level and gain measurement. Add a Boolean button Control to the front panel and change its label to "Run Measurement". On the block diagram, add a case to the event structure that handles a value change of this control, and move the control's terminal inside this case. Finally, add the *Perform Measurement* VI to this case and wire the case as shown in Figure 41.

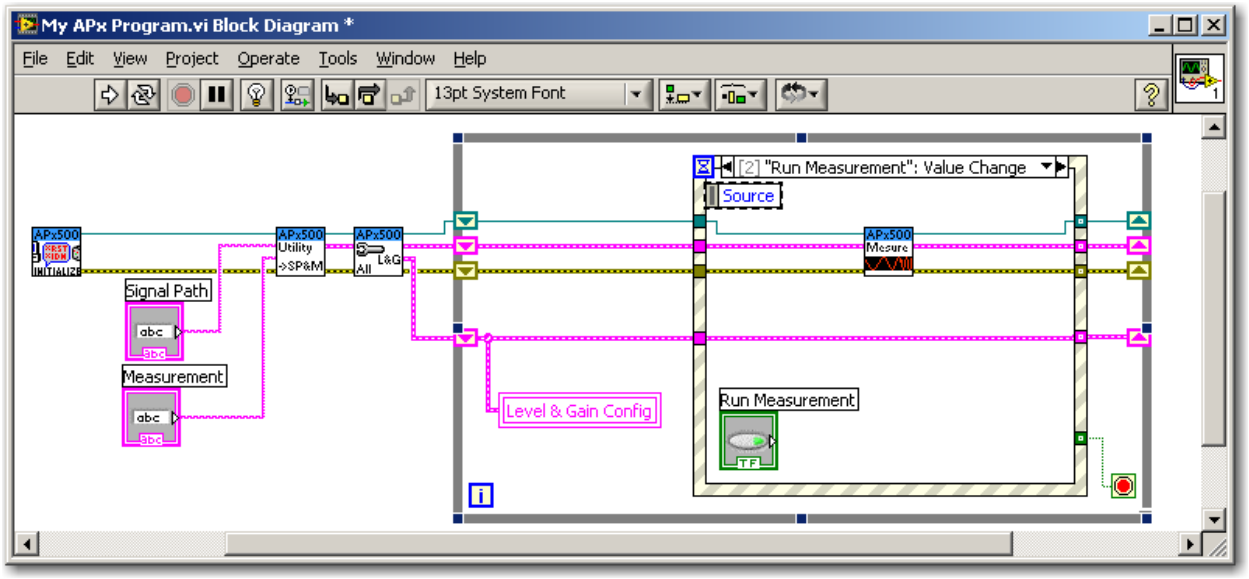


Figure 41

The front panel of the finished VI is shown in Figure 42. When you run this VI, it will get the current Generator and Signal Acquisition and Analysis settings from the APx application and

initialize the LabVIEW Level & Gain Config control to these values. Then, you can change various settings in the LabVIEW control, and pass the settings to APx by clicking the Update APx Settings button. To try it, before running the VI, change a few of the Generator settings in APx. Once you run the VI, it will update the Level & Gain Config control to the current APx settings. Next, change some settings in LabVIEW and then click the Update APx Settings button, to change the settings in the APx500 application. Finally, you can click the Run Measurement button in LabVIEW to make the APx500 application run the Level & Gain measurement.

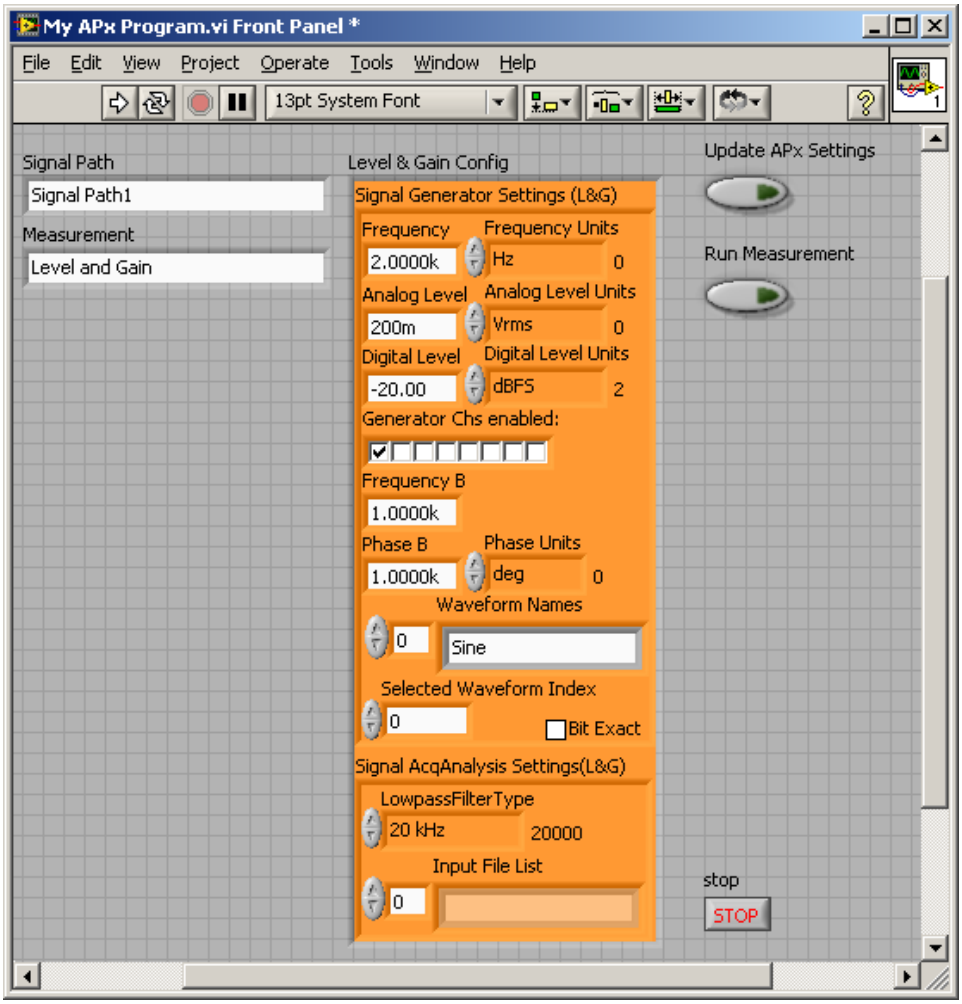


Figure 42

Note that if the Output Connector is one of the analog types (Analog Unbalanced or Balanced), when you run this VI, it sets the Digital Level control to 0.00 and its units to FS. Furthermore, if you change the Digital Level control and again click the Update APx Settings button, it again resets the Digital Level to 0.00 and the units to FS. This is because the Digital Level setting has no meaning in the context of a signal path with an Analog Output. Therefore, you should be careful to ensure that the context is correct for the controls you are using. For example, if you change the Signal Path from LabVIEW (to be discussed later in this document), be sure to *get* the measurement configuration settings after doing so.

Generating Arbitrary Waveforms with the LabVIEW Driver VIs

In the APx500 application, by default the Waveform type is Sine. This refers to the Sine generator built in to APx. Depending on the signal path setup, many of the measurements also allow other built in waveforms, such as Square waves, Split Sine waves or Split Phase, and arbitrary waveforms using imported .wav files. To add an arbitrary waveform to the project, the user selects the Browse for file… option in the Waveform list box control (Figure 43, left) and selects a waveform file from a disk drive on the PC. Once waveforms have been added to the APx project, they are available for selection from the list box (Figure 43, right). To revert back to the Sine generator, the user simply selects Sine in the list box.

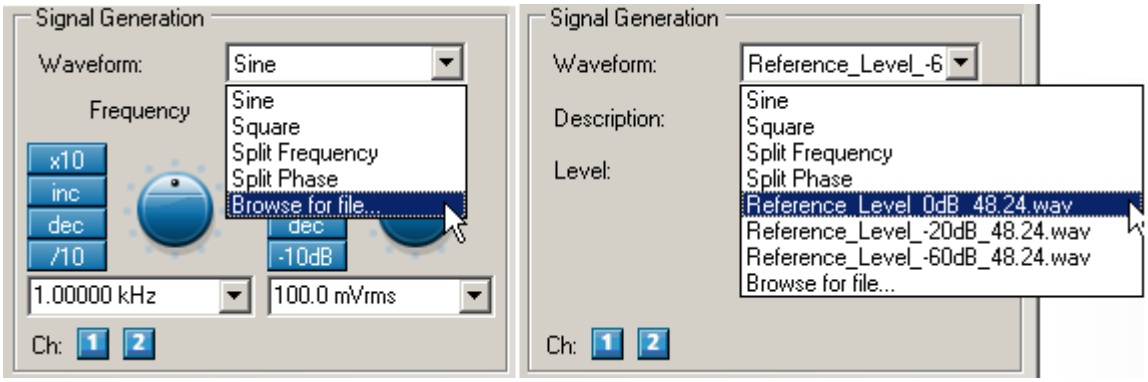


Figure 43

In the LabVIEW driver VIs, this waveform functionality is accessed using the controls shown in Figure 44, which are part of the Signal Generator Settings cluster. In the LabVIEW Driver, the Waveform Names control is an array containing the names of all the waveforms loaded into the APx project, and the Selected Waveform Index is the index of the Waveform Names array corresponding to the currently selected waveform. For example, the left side of Figure 44 shows how the controls would appear if the APx waveform was set to Sine (index 0), and the right side of Figure 44 shows how the controls would appear if the APx waveform was set to the 4th arbitrary waveform contained in the project (in this case the file named 7kHz-L_8kHz-R.wav). In LabVIEW, it is the Selected Waveform Index that is used to change the waveform in APx.

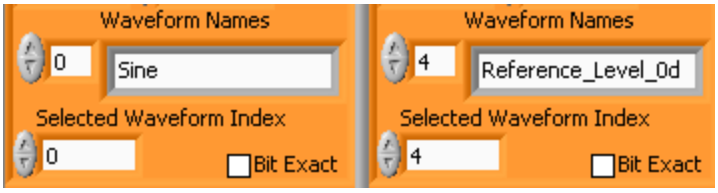


Figure 44

You can use the *My APx Program* VI to try this out: First load a few waveforms into the APx project using the Waveform – Browse for file… option shown on the left side of Figure 43. Set the Waveform control in APx to any one of the loaded files. Then run the VI. The Waveform Names control in LabVIEW will be updated to contain the list of waveforms you just loaded, and the Selected Waveform Index will correspond to the index of the selected waveform. To change the Waveform in APx, change the Selected Waveform Index to a different number, and click the Update APx Values button. The APx waveform selected will change accordingly. To change the

Generator back to Sine, change the Selected Waveform Index in LabVIEW to 0 (zero) and click the Update APx Values button.

Note that there is no provision in the APx API for adding waveforms to a project; they can only be added to a project from the APx500 application's UI.

Measuring .wav Files with the LabVIEW Driver VIs

When the Input Connector in Signal Path Setup in APx500 is set to File (Analog Units) or File (Digital Units) a button labeled "File List…" appears in the Signal Acquisition and Analysis section of the measurement controls area. When clicked, this button allows the user to specify a list of .wav files to be measured (Figure 45).

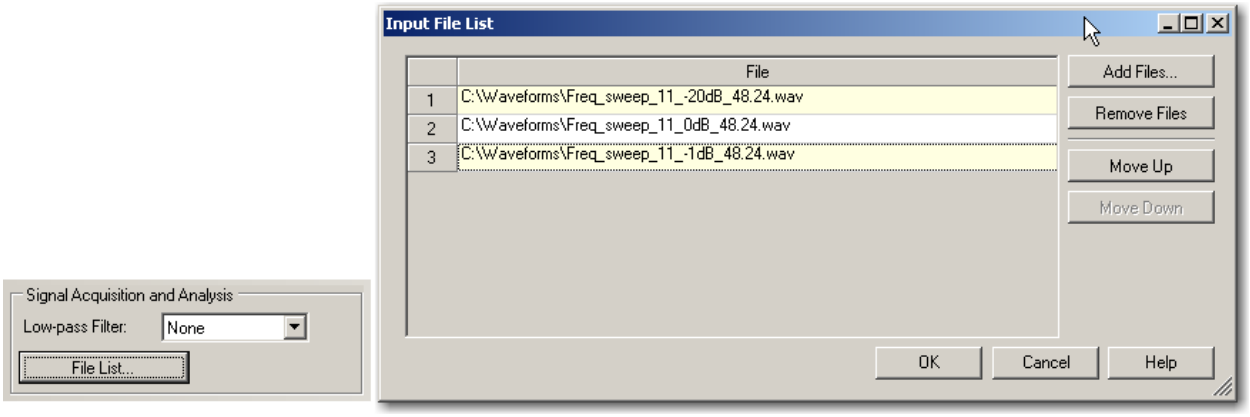


Figure 45

In the LabVIEW Driver VIs, this list of .wav files is handled as a string array labeled "Input File List" in the Signal Acquisition and Analysis Settings cluster control (Figure 46). When LabVIEW gets the settings, this string array will be loaded with strings such that each element of the array represents the full name and path of one file in the APx file list. The LabVIEW String to Path function can then be used to convert these strings to LabVIEW file paths. To remove a file using LabVIEW, string elements may be removed from the end of the array before passing the array back to APx using the set function.

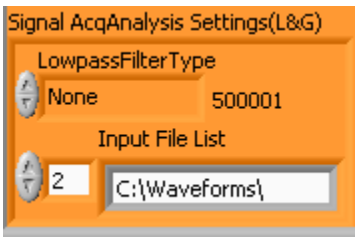


Figure 46

Recap–Configuring Measurement Settings

The above example illustrates changing the APx Signal Generator and Signal Acquisition and Analysis settings for the Level and Gain measurement. The APx LabVIEW .NET Driver VIs use this same model to change the settings of all measurements in APx. If you look at the configuration section of the VI Tree (Figure 11) you will see that each measurement has either

one or three configuration VIs. Those measurements with three VIs are ones like Level and Gain, which have both Generator settings and Signal Acquisition and Analysis settings. In this case, there is a *GetSet Generator* VI, a *GetSet Signal Acquisition & Analysis* VI and a *GetSet All* VI. This allows LabVIEW user to *get* or *set* either of the Generator settings, the Signal Acquisition and Analysis settings, or both.

Measurements with only one configuration VI are those which in APx have only Generator settings (e.g., Frequency Response) or Signal Acquisition and Analysis settings (e.g. Noise), but not both (Figure 47). These VIs should be used exactly as described above. You should *get* the current settings from APx before changing any parameters, and then do a *set* to update APx. In addition, the *set* VIs always do a *get* after setting, to ensure that they return the correct state of APx.

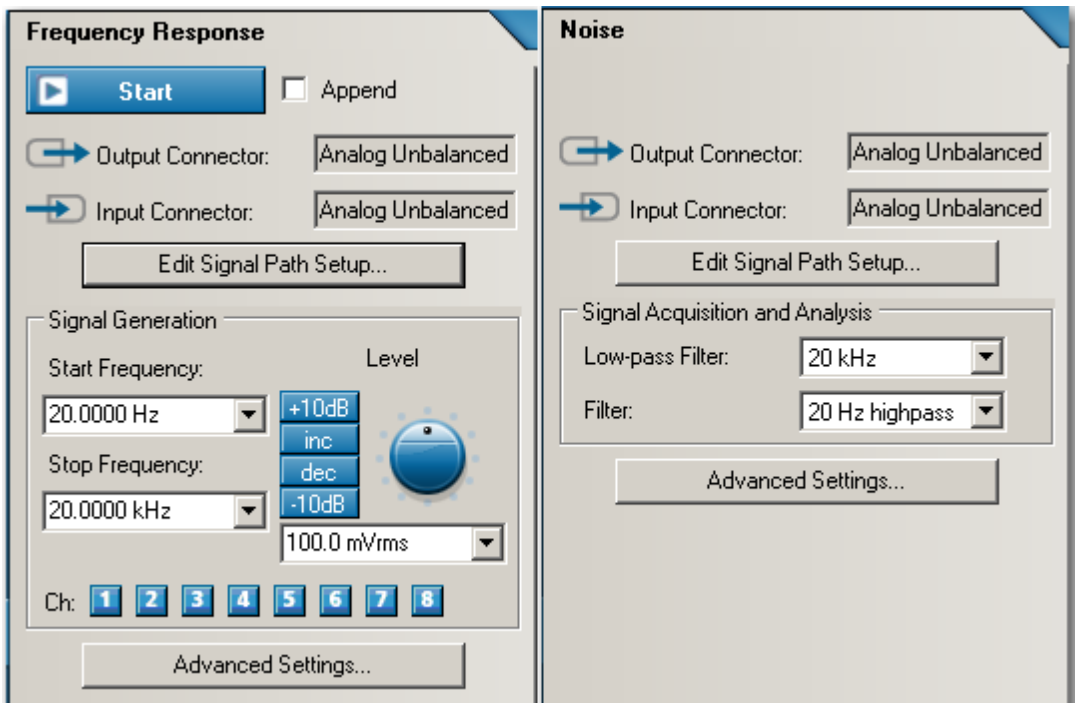


Figure 47

About the Orange Colored Controls

You may be wondering why the LabVIEW Driver VI cluster controls introduced so far in this document have been orange colored. It's not because we like the color orange. Rather, the controls colored orange in the driver VIs are a special type of LabVIEW custom control known as a Type Def. (short for Type Definition). In LabVIEW, a Type Def. control is a kind of "master" control.

When a Type Def. control is used in a collection of VIs, it is much easier to maintain the code because if you alter the Type Def, any VIs containing that control will be automatically updated as well. This is a handy feature. For example, there is a Low-pass Filter control that is used in many of the VIs in the driver collection. Suppose that the number of VIs it is used in is fifty. If a Type Def. was not used for this control, when a new type of Low-pass filter is added to APx in the future, all fifty of the VIs containing this control would have to opened and modified

individually. Since the control is a Type Def., it only needs to be changed once, and the fifty dependent VIs will be updated automatically. The orange color is used simply to make it obvious that a custom control is a Type Def.

It is easy to change the orange color of controls used in the driver VIs. First, you need to disconnect the control from its Type Def. To do so, right-click on the edge of a cluster control and select Disconnect from Type Def. You will be prompted with a dialog box to confirm this operation. Note that many of the cluster controls contain sub-controls that are also Type Defs. Therefore, you may need to disconnect several controls from their Type Defs to completely remove the orange color. Figure 48 illustrates the process for one of the Type Defs in the Level & Gain Config cluster, and the final result when the color of every orange control has been changed.

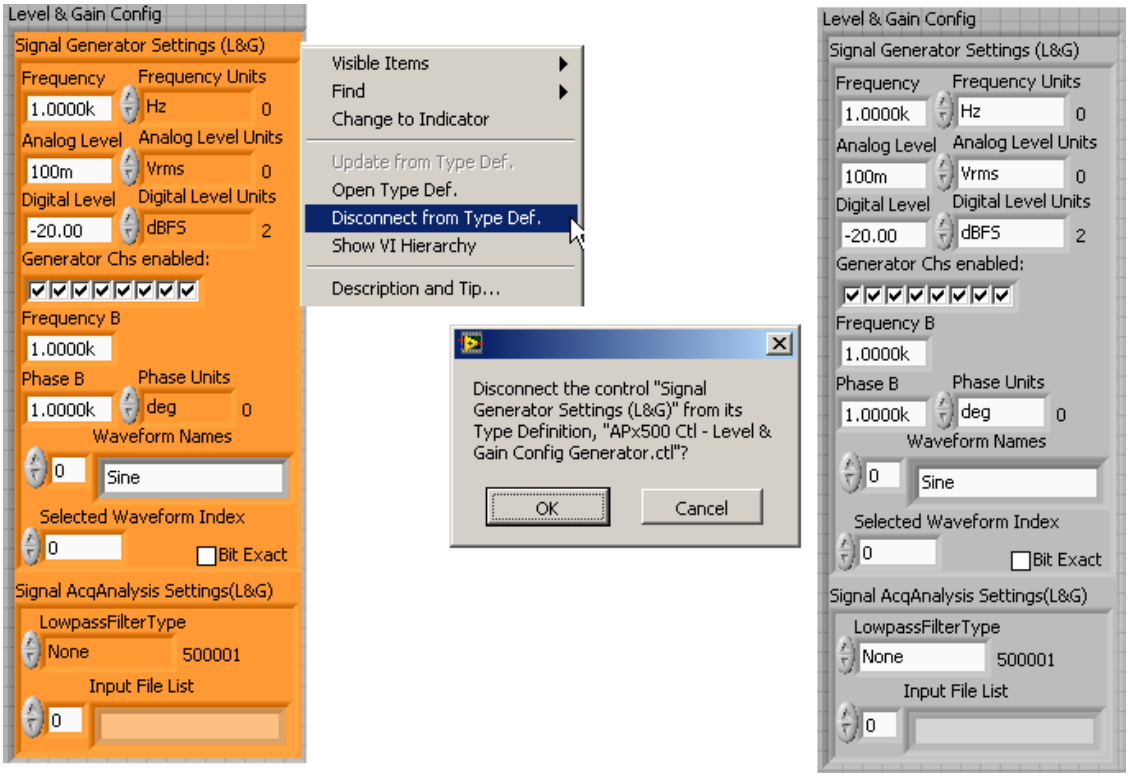


Figure 48

Working with Cluster Control Subsets

Most users will not want to set all of the controls included in a large control cluster like the Level & Gain Config control. For example, suppose you are only using the Analog Unbalanced Output Connector in APx, and all you want to do is control the generator level and frequency. In this case, most of the controls in the Level & Gain Config cluster are not necessary. To handle this scenario, consider customizing controls.

For example, in the *My APx Project* VI, let's create a control that will allow us to change only the analog generator level and frequency. Proceed as follows.

1. Since we are only changing the generator settings, we can use the *Level & Gain GetSet Generator* VI instead of the *GetSet All* VI. Right-click on the *GetSet All* VI on the block diagram, and use the Replace menu to change the VI (Figure 49).

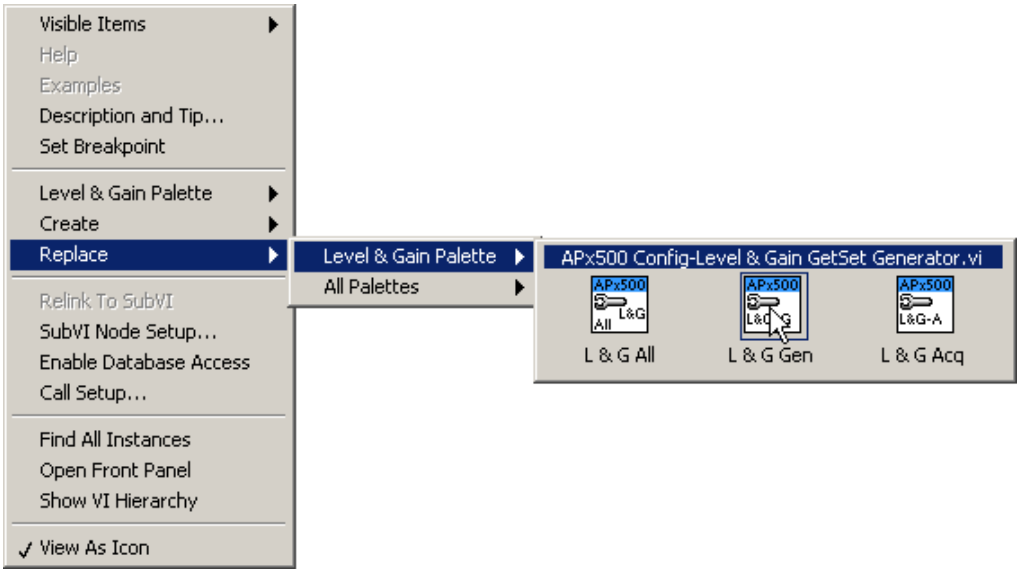


Figure 49

2. Disconnect the Level & Gain Config control from its Type Def. (if is not already disconnected) and disconnect the Signal Generator Settings (L&G) cluster from its Type Def.
3. Drag the Frequency and Analog Level controls out of the Level & Gain Config control cluster.
4. Delete the Level & Gain Config cluster, so that only the Frequency and Analog Level controls remain in its place (Figure 50 and Figure 51).

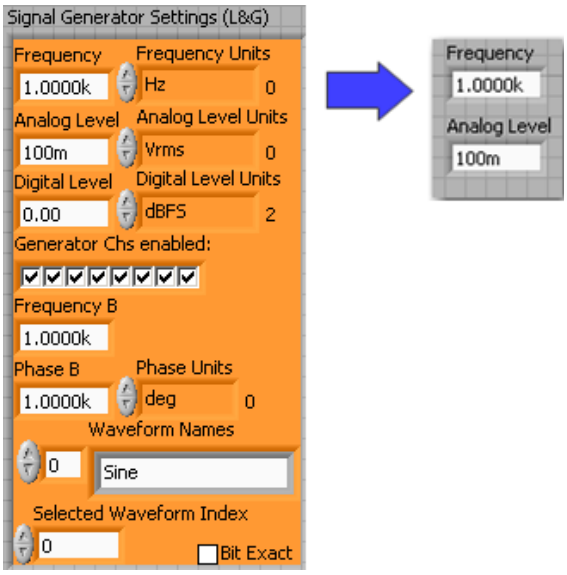


Figure 50

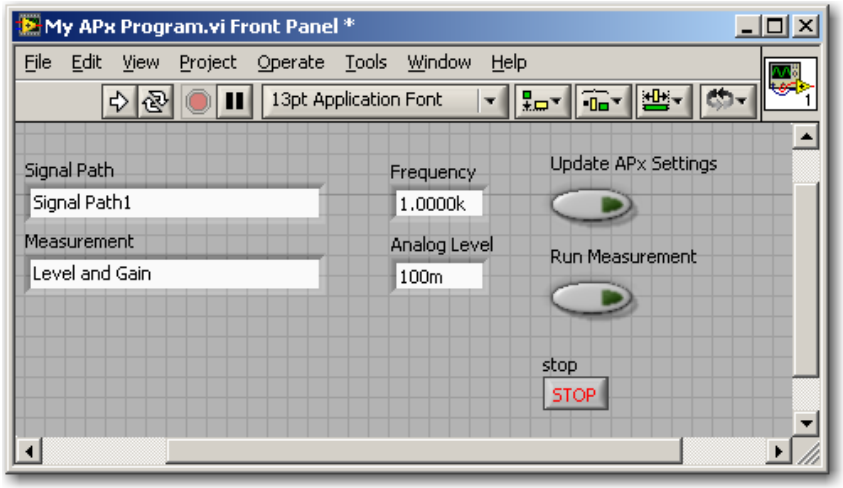


Figure 51

5. Finally, rewire the block diagram as shown in Figure 52. Note the use of the Unbundle by Name function outside the Event Structure, and the use of the Bundle by Name function. These functions are used to get the values of only the Frequency and Level controls from the Signal Generator Settings (L&G) output cluster, and to pass them back in to the corresponding input cluster.

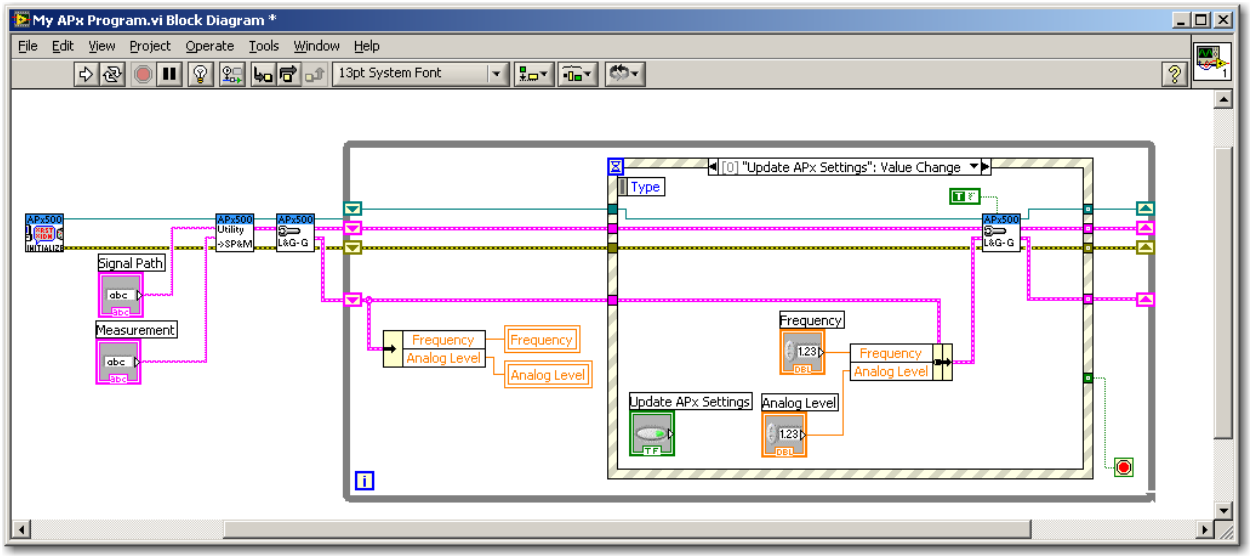


Figure 52

The *My APx Program* VI as modified to allow the user to change only the generator frequency and level of the Level and Gain measurement is now ready to run. If you run it, you will see that it functions as it did before.

The APx LabVIEW .NET Driver VIs use cluster controls extensively. The best way to get individual data items out of a cluster or into a cluster is by using the Bundle by Name and Unbundle by Name functions in LabVIEW. Figure 53 (left) shows the Unbundle by Name function being used to extract specific elements of the Signal Generator Settings (L&G) cluster.

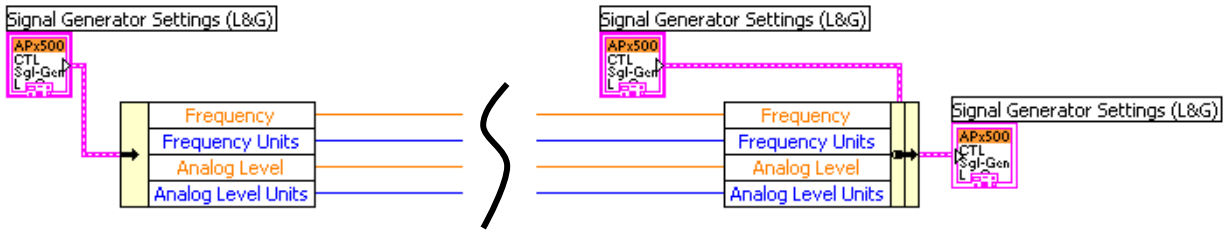


Figure 53

Figure 53 (right) shows the Bundle by Name function being used to replace 4 specific elements of a Signal Generator Settings (L&G) cluster.

Accessing Primary Measurement Results with the LabVIEW Driver

Although the examples to date have illustrated running measurements in APx, we haven't yet considered how to work with the results of APx measurements. To be of value, an external program that controls the APx500 analyzer must be able to access the measurement data. This section will focus on accessing APx primary measurement results (results that normally appear when you add a measurement to the navigator) with the LabVIEW driver. The following sections discusses derived measurement results

First, let's review what measurement results look like in APx. Create a new project in APx using the default project template, and add a Stepped Frequency Sweep to Signal Path1. If you expand the branch of the Navigator tree containing the Stepped Frequency Sweep measurement, you will see a number of objects within this branch named Level, Gain, Relative Level, etc. These are measurement Results of the Stepped Frequency Sweep measurement. They can also be selected by clicking on the appropriate icon in the window beneath the Graph in APx (Figure 54).

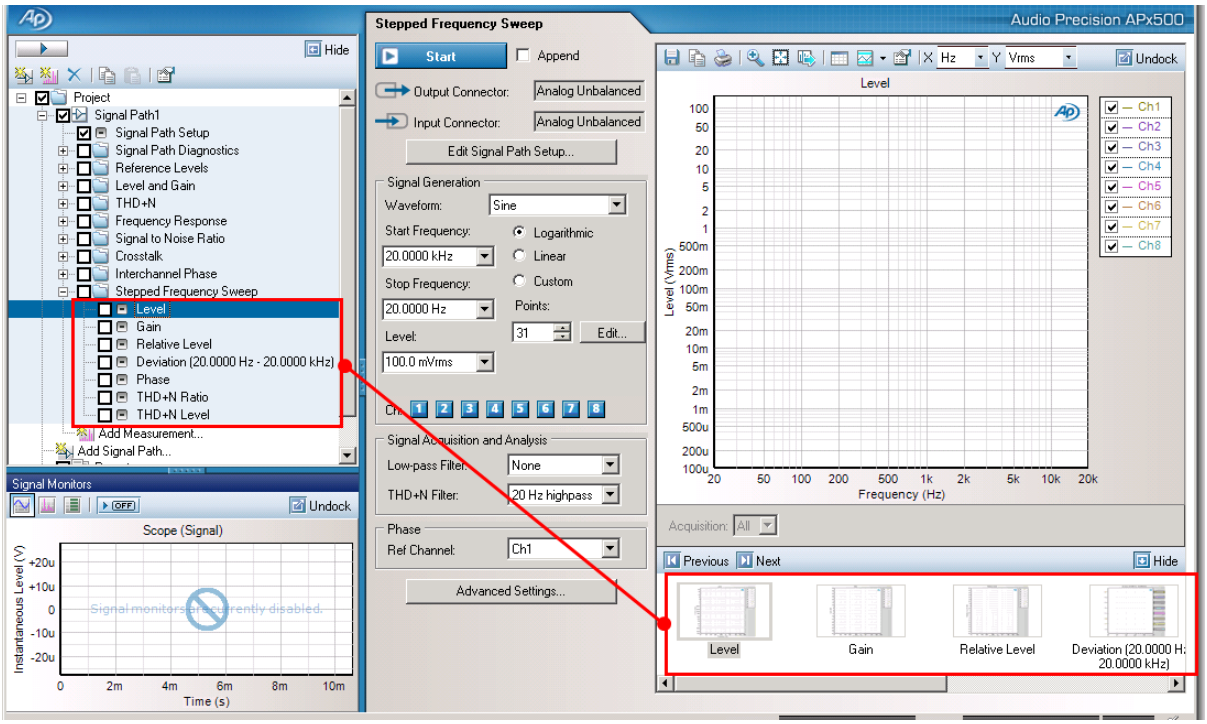


Figure 54

In the case of the Stepped Frequency Sweep measurement, there are seven results available. If you run a Stepped Frequency Sweep measurement and browse through the results, you will notice that six of the seven results are displayed as an XY graph (e.g., Level is displayed as a graph of Level versus frequency). These are referred to as XY type results. One of the results in the collection - Deviation - is displayed as a bar graph with one bar per channel Figure 55. These latter results, which consist of a single value, are referred to as Meter type results.

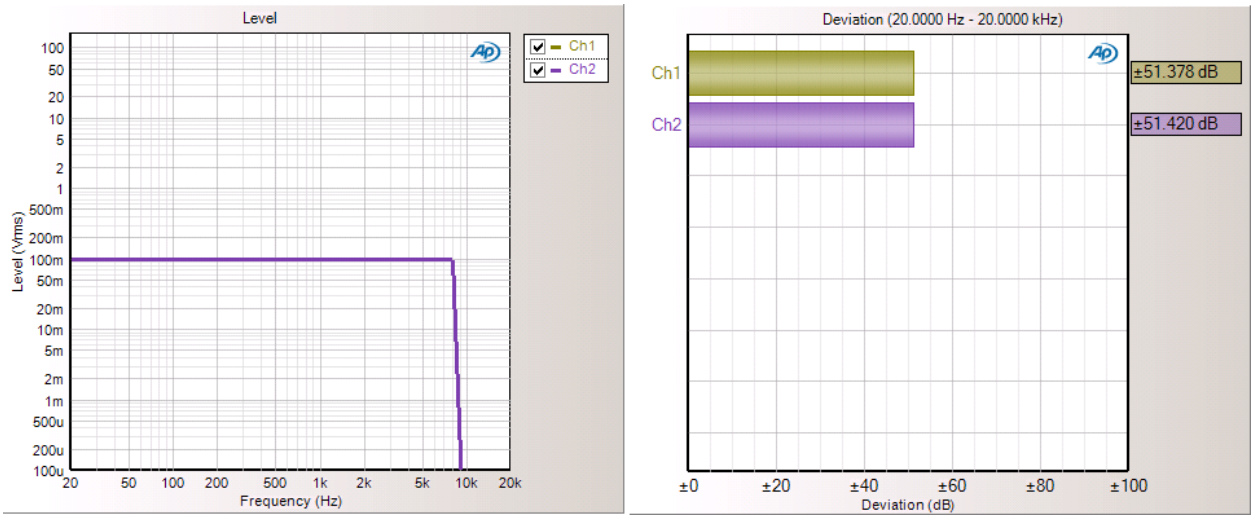


Figure 55

Each measurement has a different results collection - some have all Meter type results, some have all XY type results, and some have a combination of the two result types.

Now let's look at how the LabVIEW driver handles measurement results.

Open the *APx500 Example-Simple Meter Measurement* VI (available from the Examples sub-palette of the driver palette) and examine its front panel. Ignore the orange color of the controls. This simply denotes that they are still associated with their Type Defs. This VI is set up to configure and run a Level and Gain measurement located in Signal Path1 (one of the defaults for a new project file created with the default template). Notice that the Selected Measurement Index is set to 3 (the position of the Level and Gain measurement in Signal Path1.

The Level & Gain Config control as set in the project will set the analog generator to 100 mVrms or -20 dBFS at 1 kHz, and enable all generator channels. Configure the APx500 application to use two Input Channels (in Signal Path Setup). Now run the VI and observe the contents of the Measurement Results Cluster (Figure 56). Note that it now lists the Path Name as Signal Path1, and the Measurements array inside the Measurement Results cluster has one element - with Measurement Name field containing "Level and Gain". Inside the Data cluster are two more arrays named "XY Results" and Meter Results". Note that the XY Results array is empty (this makes sense, because Level and Gain has no XY type results). Also, the Meter Results array has two elements: one for the Level result and one for the Gain result. As shown, clusters in the Meter Results array contain an indicator showing the result name, the units, and whether all channels passed upper and lower limits.

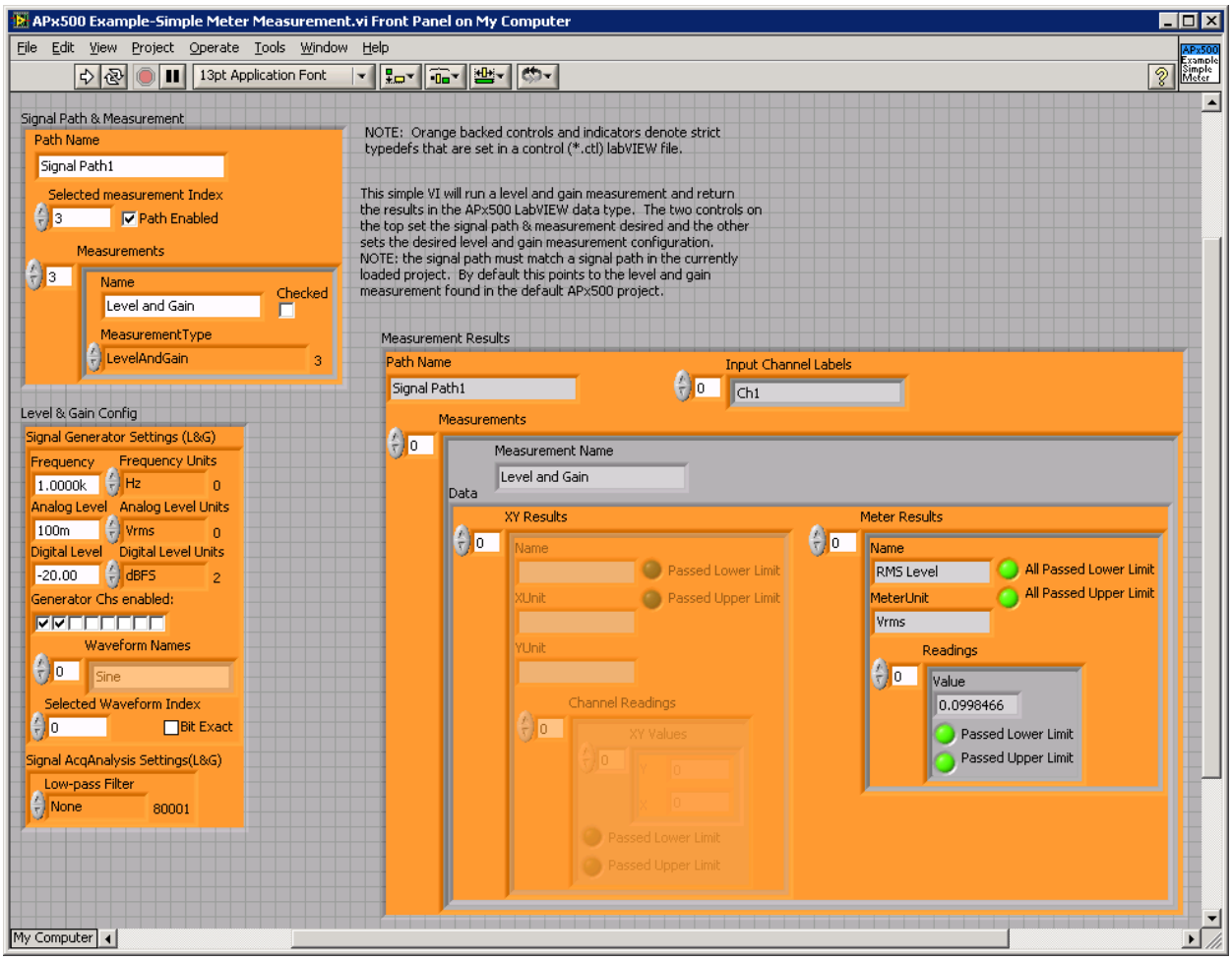


Figure 56

If you change the index of the Meter Results array, you will notice that there are two elements in the array, corresponding to the two results available (Level and Gain). Inside the cluster is an array called Readings, which in turn contains a cluster of result Values, and indicators Passed Upper Limit and Passed Lower Limit (Figure 57). Note that the number of elements in the Reading array corresponds to the number of Input channels selected in APx (in this case two).

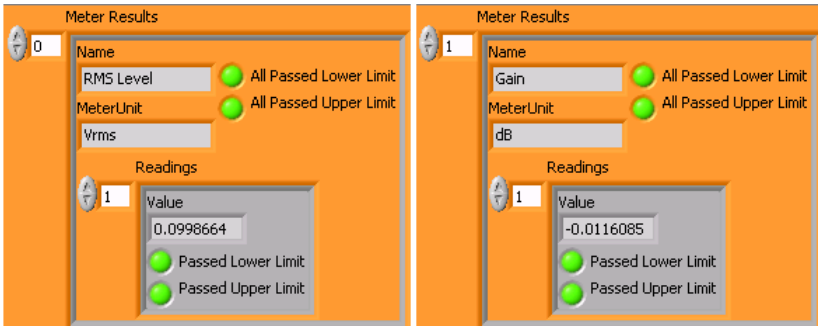


Figure 57

At first glance, the Measurement Results cluster appears complicated. However, if you study it, you will find that the data is organized in much the same fashion as the measurements results are

organized within the APx500 application. Furthermore, it is a very efficient scheme for getting the measurement data from APx, because all of the results from one measurement are contained in a single cluster that can be passed to subVIs in a single wire (Figure 58). In fact, the Measurement Results cluster can hold the results for all the measurements in one signal path. This will be demonstrated later, with an example that runs a Sequence in APx.

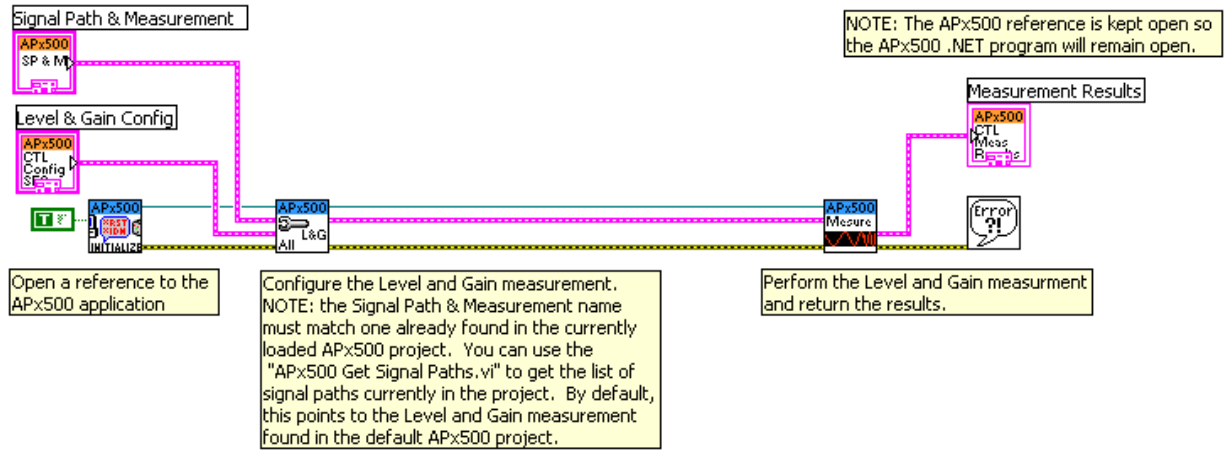


Figure 58

Note in Figure 58 that there are only three VIs needed to open a reference to APx, select and run a Level and Gain measurement, and get back all the data for that measurement.

There is a similar example in the driver VI collection named *APx500 Example-Simple Sweep Measurement.vi*. This example runs a Stepped Frequency Sweep and returns the Measurement Results cluster. If you run it, you will see that the XY and Meter results returned correspond to those available for this measurement in APx.

There are two driver VIs in the collection that simplify getting Meter results and XY results from the Measurement Results cluster. Their context help is shown in Figure 59.

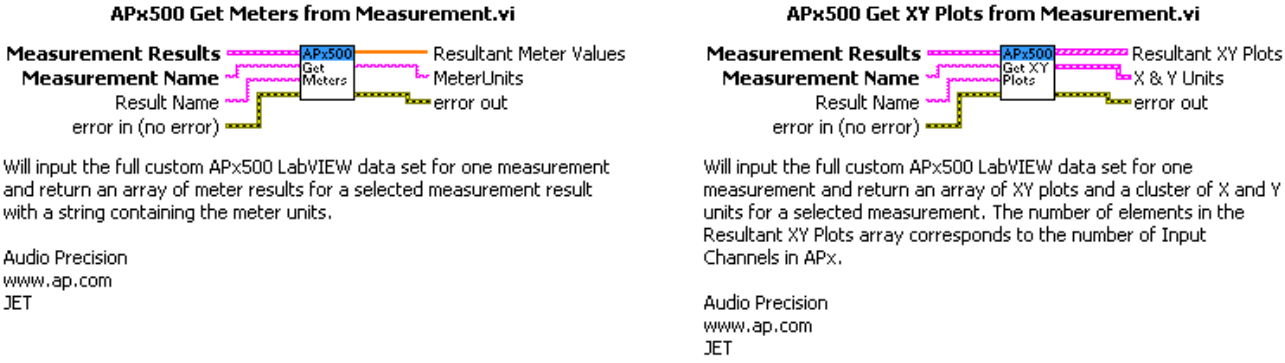


Figure 59

For an example of how to use the above two VIs, see the example VI named APx500 Example-Simple Data Results, available from the Examples sub-palette of the driver menu Palette.

Handling Dynamic Measurement Results

Measurement results are dynamic, meaning that they can be added, deleted, or renamed. Furthermore, *derived* measurement results, which are computed from a source result, may also be added, deleted, or renamed. When using the LabVIEW driver to access APx data, the main consequence of this is that you need to ensure that the desired result exists in a measurement, and that you are using the correct result name. Otherwise, the APx API will throw an exception.

In Sequence mode, when a single measurement is run, this is equivalent to right-clicking on a measurement in the APx software and selecting Start Selected Measurement. Once the measurement is complete, APx builds the Sequence results collection, which contains all the results that were checked before the measurement was run. If you use the VIs in Figure 59 to attempt to access meters or XY result data, the sequence collection will not contain the desired result if the result name does not match, the result was deleted from the measurement, or the result's checkbox was unchecked in the navigator. To handle this case, a custom error (number 7001) has been created, as shown in Figure 60.

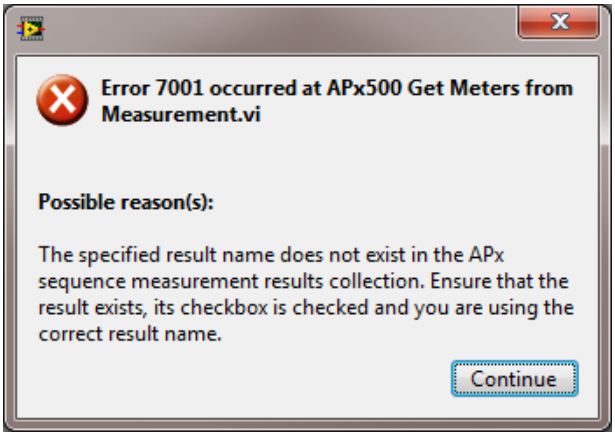


Figure 60

Returning All Data Points

In Sequence mode, measurement results are added to the Sequence Data Buffer every time the sequence is run. Whether XY result data is Same as Graph or All Points depends on the Result Specification that is set for each measurement result before the sequence has been run. The following VIs are affected by the Result Specifications:

- APx500 Sequence – Get Measurement Results
- APx500 – Sequence Run All Measurements
- APx500 Sequence – Perform Measurement

The following VIs always return All Points:

- APx500 – Run Continous Sweep and Get Acquired Waveform
- APx500 – Run Signal Analyzer FET All Points
- APx500 – Run Signal Analyzer Scope All Points

The default Specification is Same as Graph, so in order to retrieve All Points, the settings must be changed. This can be done in the Export Graph Data dialog box of each measurement (note that you need to already have result data, and need to export to a file in order to save the Specification changes).

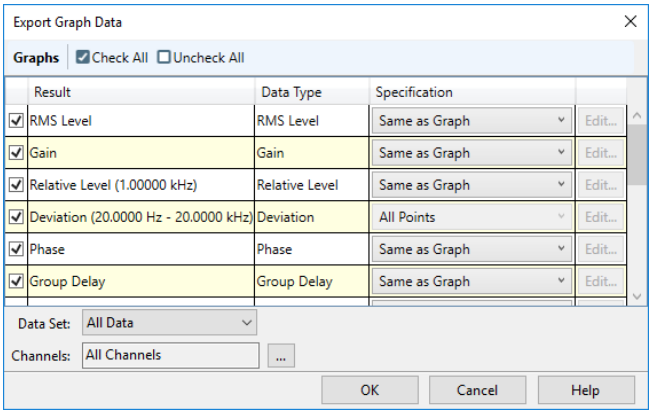


Figure 61 Export Graph Data, Same as Graph

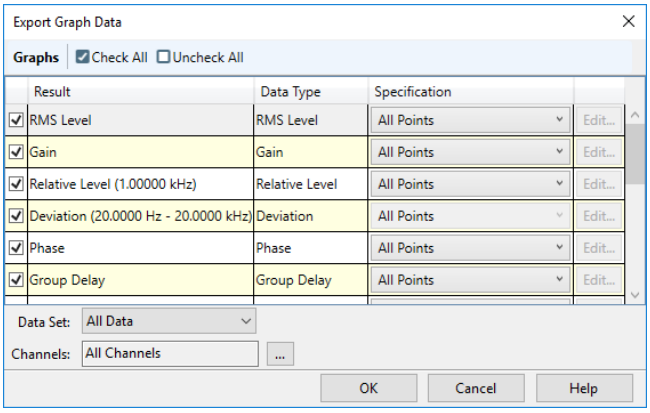


Figure 62 Export Graph Data, All Points

We have also created a VI, APx500 Data – Set All Points, that traverses every Signal Path, Measurement, and Result in a project and sets the Specification to All Points. This VI only needs to be run once and then the project may be saved to preserve the Specification settings.

Note the following pointers about saving data:

- If the number of points is low, then Same as Graph and All Points may return the same data set.
- The Bench mode VIs, APx500 Bench – Get Measurement Results, and APx500 Bench – Export Results, always get All Points.
- When using a .NET invoke node to retrieve data directly from a measurement (instead of from the Sequence Data Buffer), use GetAllXValues and GetAllYValues to get All Points.

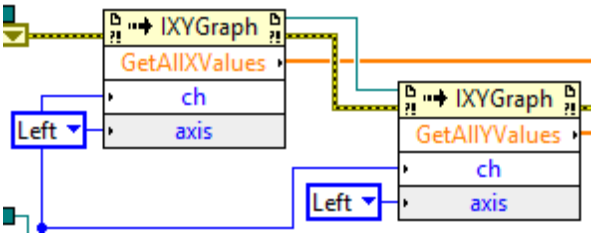


Figure 63

- When using a .NET invoke node to export data from a measurement to a file, use All Points as the exportSpecification argument.

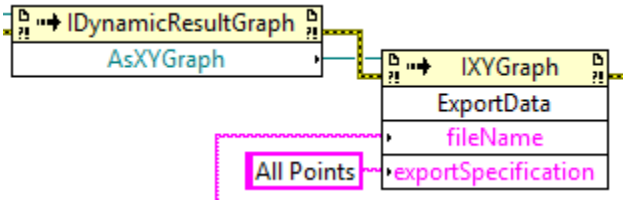


Figure 64

How the Driver VIs Handle Measurement Data Units

Currently, driver VIs are configured to return measurement data in the same units as currently set within the APx500 application. To use different units for a measurement result, you must change the units displayed for that result in the APx500 application.[3]

Running an APx Measurement Sequence

Included in the driver collection is a VI named *APx500 Example-Run Project Sequence.vi*. This example is equivalent to clicking the Run Sequence button in APx. It will run all checked measurements within all checked signal paths and return all measurement results for the entire sequence in a LabVIEW array of Measurement Results. Figure 65 shows the Measurement Results array that resulted from a Sequence being run on an APx project with two signal paths. The first checked measurement in the second signal path (index zero) was a Continuous Sweep measurement. Note that the elements in this array of clusters are the same as Measurement Results cluster returned by the *Perform Measurement* VI. In this case, the array contains one element for each signal Path in the sequence. This is a very efficient way to get an entire sequence worth of measurement data from APx to LabVIEW. And as shown in Figure 66, the LabVIEW code required to get all this data is very simple.

[3] Note the API provides the ability to change units programmatically, but this is not supported in this release of the APx LabVIEW Driver.

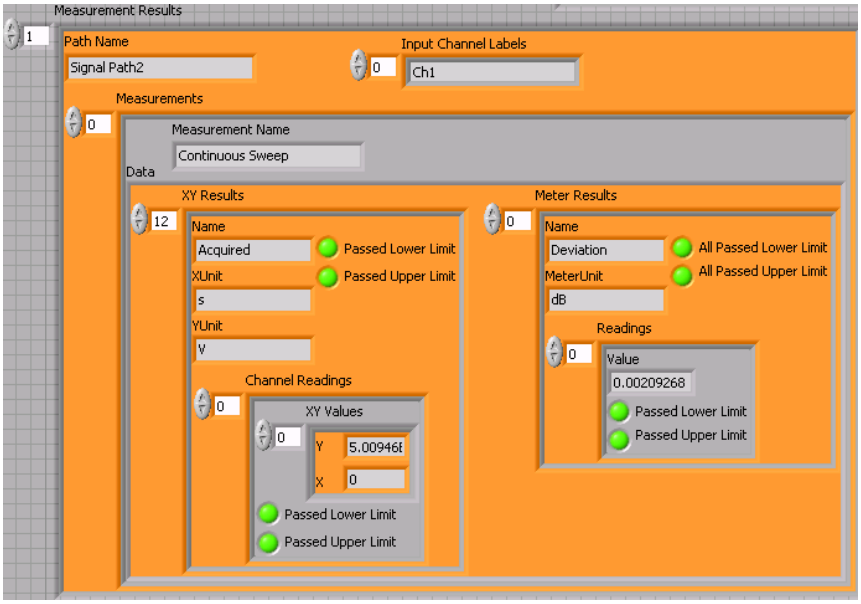


Figure 65

Another variation on running a sequence is illustrated in the example named *APx500 Example - Run Sequence & Save PDF Report.vi*, shown in Figure 67. This example was designed to look more like a User Interface (there are no orange colored controls). It runs a sequence and optionally exports the APx report to an external file (PDF, HTML, RTF, xls or text). This example illustrates the use of three utility VIs included in the driver collection: one to open a project file, one to set the visibility of the APx500 application, and one to turn the APx signal monitors on or off. See the context help for these VIs in Figure 68. Note that turning the signal monitors off can save significant CPU resources, especially with high channel counts.

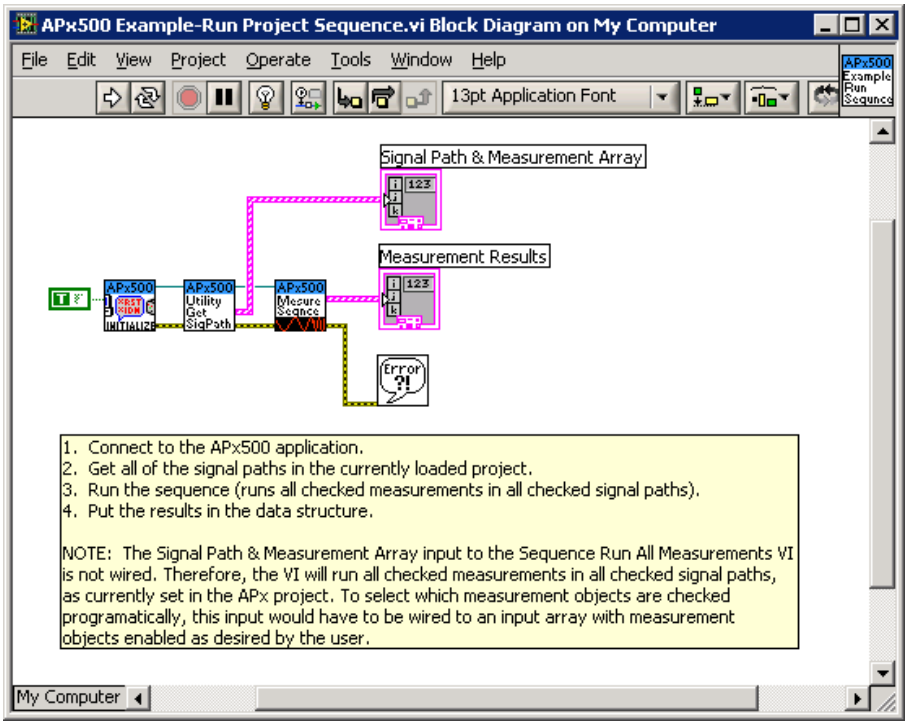


Figure 66

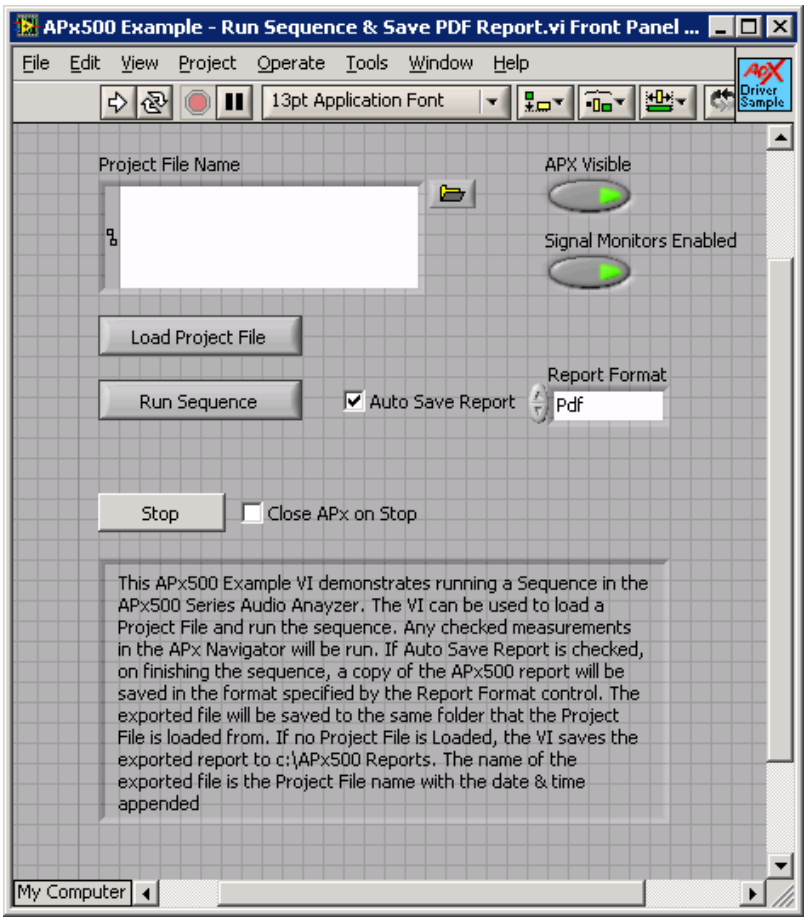


Figure 67

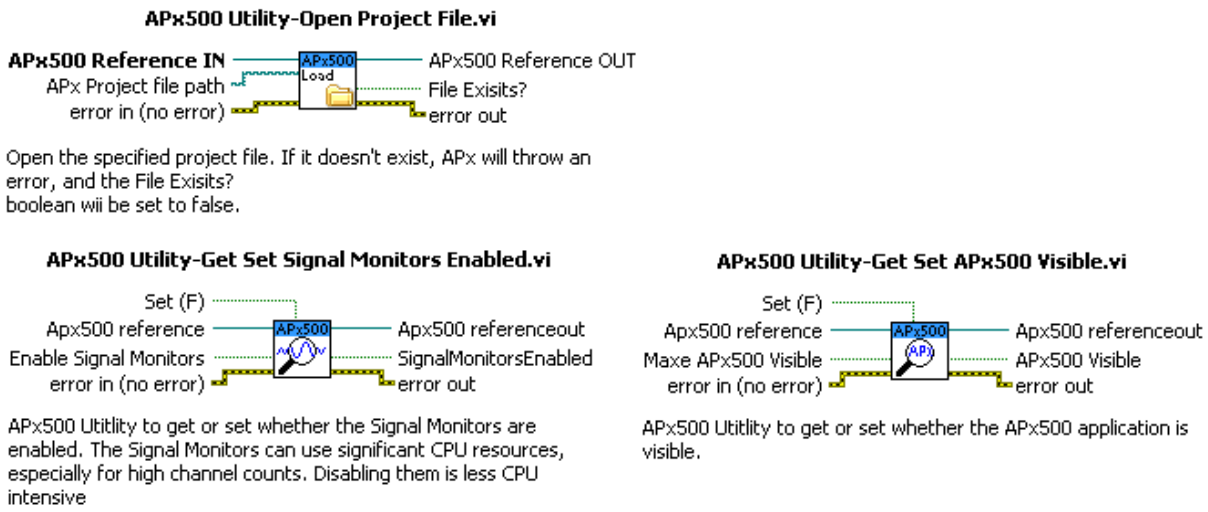


Figure 68

The User Interface Example

One of the examples included with the driver is a full-featured user interface. Please see the VI named *APx500 Example - User Interface.vi*, whose front panel is shown in Figure 69. This VI allows the user to load any APx project file. It then gets a list of all Signal Paths and measurements included in the APx project, and adds them to the Selected Signal Path and the Measurement Selection controls, respectively, on the Test Configuration Panel of the VI. While the VI is running, whenever the user changes the Measurement Selection, the configuration controls for that measurement become visible on the panel to the left of the Test Configuration panel. Here, the user can change any of the measurement configuration settings.

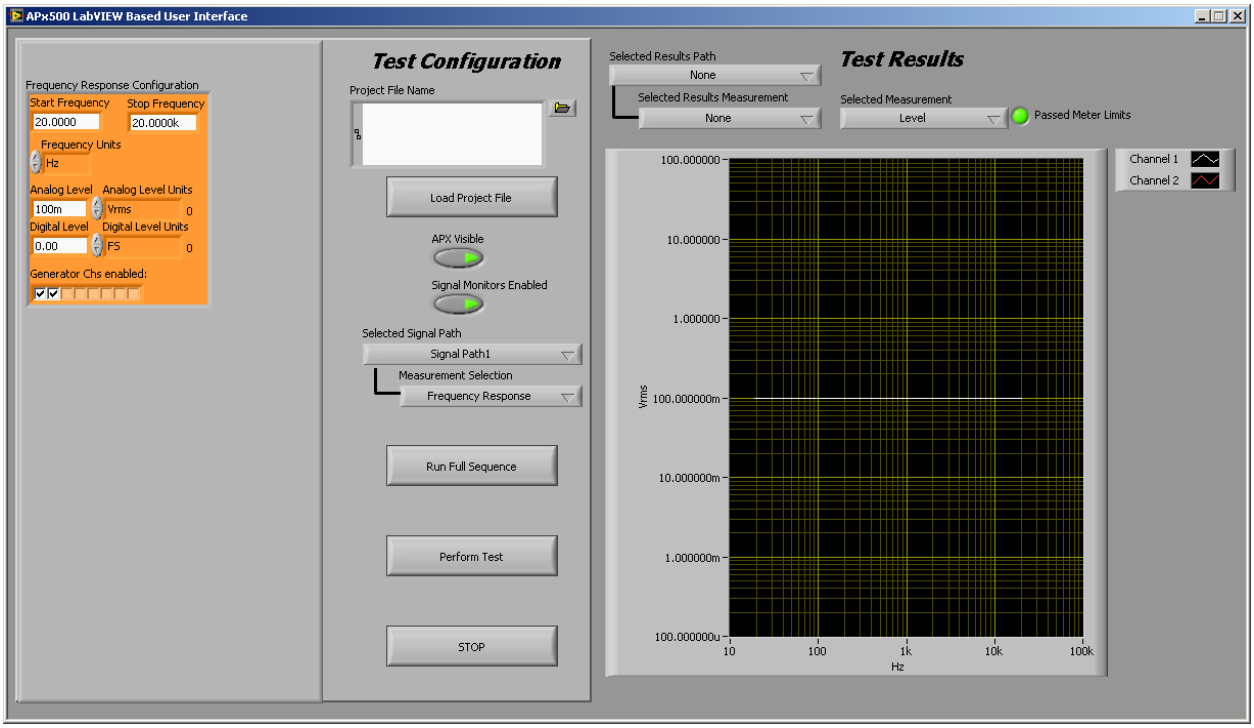


Figure 69

The Perform Measurement button on the Test Configuration panel will cause the APx500 application to run the selected measurement. After the measurement is complete, on the Test Results panel, the user can select which measurement results to display. XY graph type measurements are displayed as a graph, and Meter type results are displayed as a list.

This example VI can also be used to run all checked measurements in the sequence. If the sequence is run from the VI, all of the sequence measurement results will be available for browsing in the Test Results panel of the VI.

The User Interface example is not meant to replace the APx500 application's UI, but it does demonstrate many of the feature available in the LabVIEW Driver for working with the APx API. You may wish to customize this example to create LabVIEW applications that suit your own needs.

Configuring the Signal Path Setup

The most convenient way to configure the Signal Path Setup in APx is to do so using the APx500 application's UI, then save the project to a project file, and open the project file from LabVIEW. However, if you want to configure the Signal Path Setup from LabVIEW, the driver does have VIs to do so. The first two VIs to consider in this regard are shown in Figure 70. One will *get* or *set* the Output Connector Type and the other will *get* or *set* the Input Connector Type. The function of these VIs is very similar to that of the measurement configuration VIs discussed earlier.

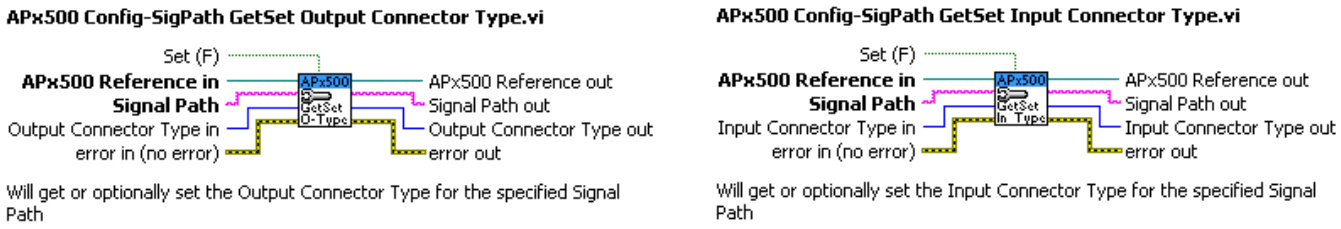


Figure 70

For an example of how to set the Output Connector Type, see the VI named *APx500 Example - Set Output Connector Type.vi.* This VI's front panel is shown in Figure 71, and its block diagram is shown in Figure 72. If you run the VI and change the Output Connector Type, this change will be reflected in the APx500 UI, if the connector type is valid for the connected instrument. If the connector type is not valid (for example, setting the control to Digital HDMI when the HDMI option is not present), APx will generate an error. The function of the *GetSet Input Connector Type* VI is very similar.

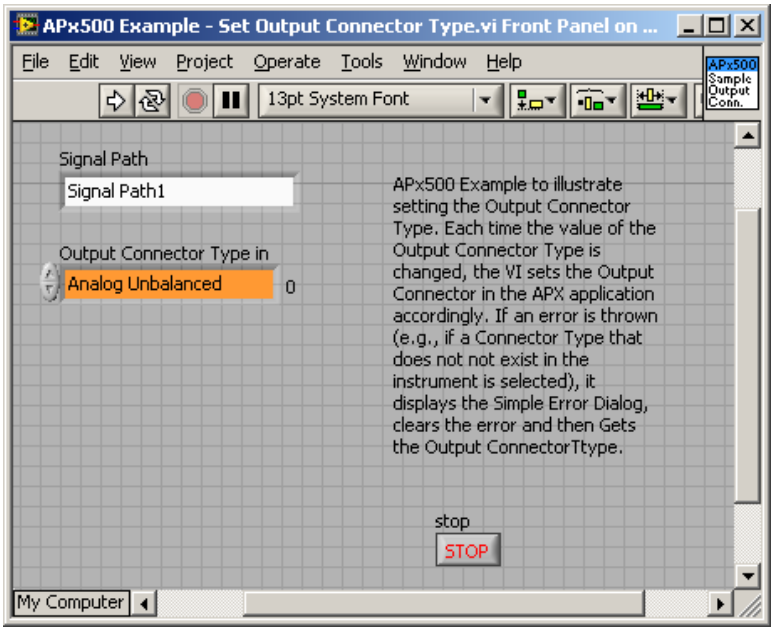


Figure 71

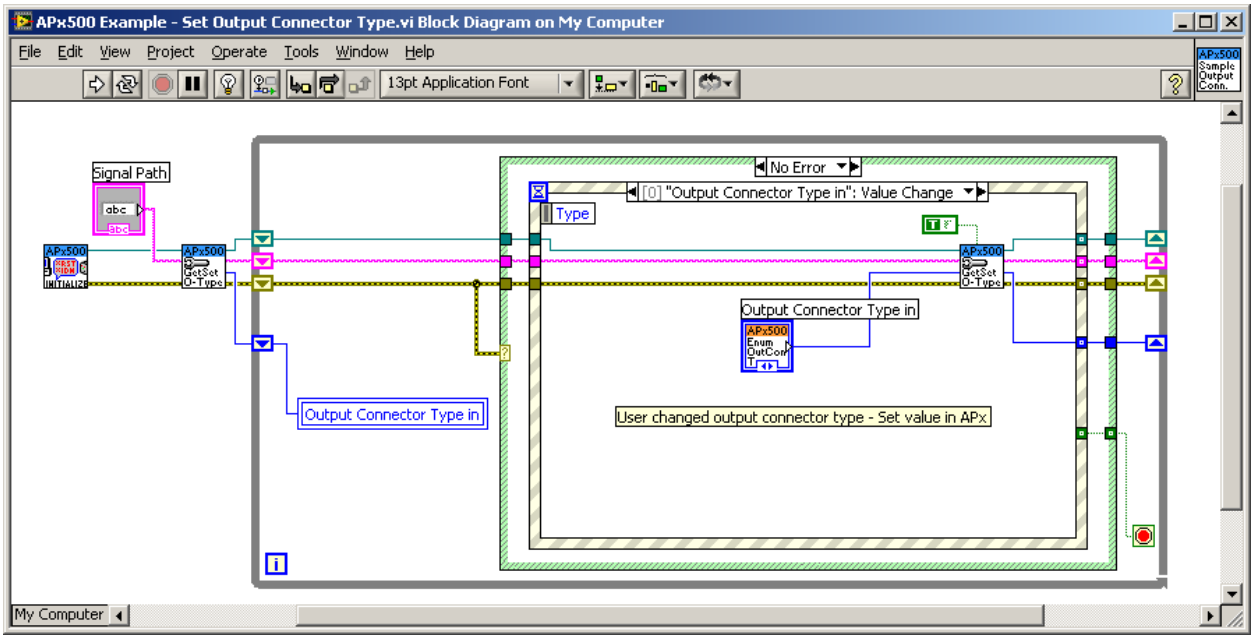


Figure 72

In the APx500 application, settings can be configured for each Input and Output Connector type. Note the Settings… button to the right of the Connector list box control in the Output Configuration area of Signal Path Setup. When you click this button, the Output Settings dialog box for the selected connector type opens as shown in Figure 73. There is a similar Settings dialog for the Input Configuration in APx500.

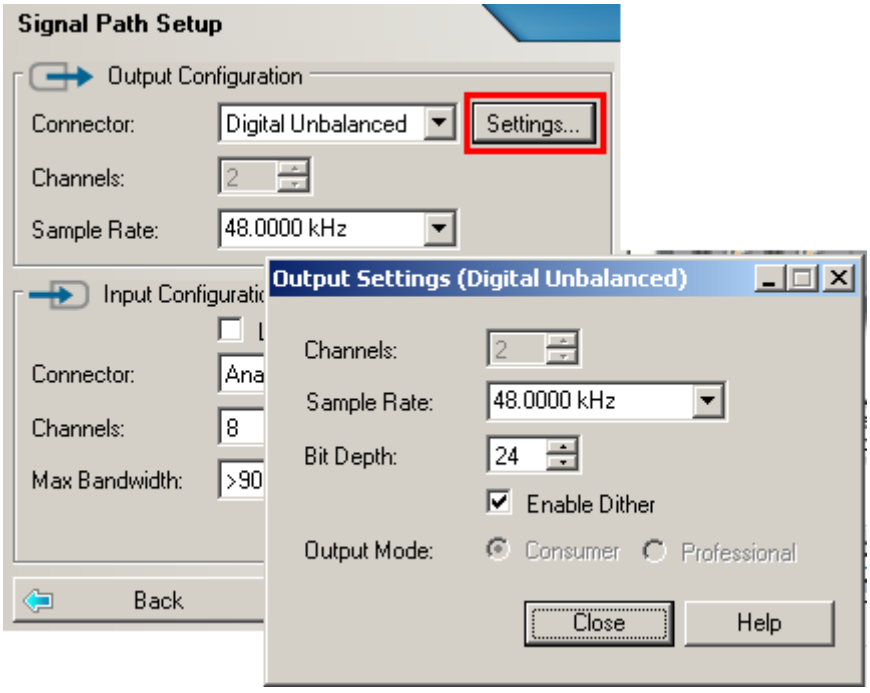


Figure 73

In Figure 73, the selected Output Connector is Digital Unbalanced. The VI for configuring these settings is named *APx500 Config-SigPath GetSet Output Digital Unbalanced*. Its context help and configuration settings control are shown in Figure 74. Note the correspondence between the LabVIEW controls in Figure 74 and the APx500 controls in Figure 73.

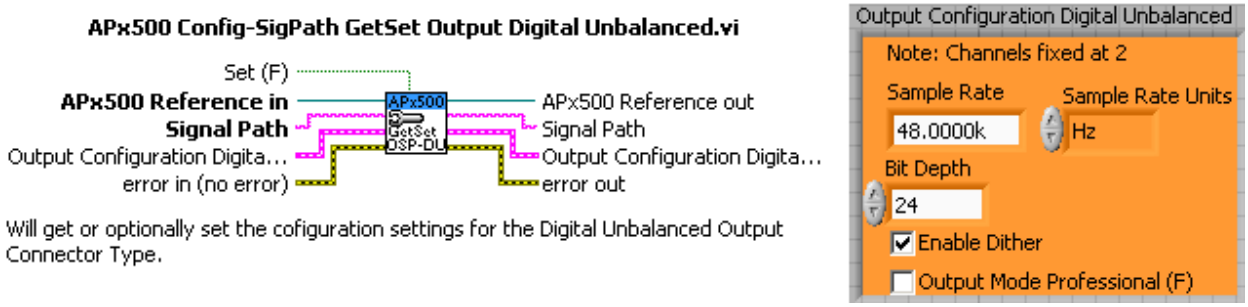


Figure 74

There are similar VIs for configuring input configuration settings and output configuration settings for all the possible instrument connector types. Note that the VIs for Digital Serial do not provide all the settings, but allow you to select a Digital Serial settings configuration file where all the settings can be stored. In addition, there is no VI for the None (External) output connector type, because in this case there are no settings to set.

Figure 75 shows an example VI from the driver collection named *APx500 Example - Configure Digital Unbalanced Output*. This example illustrates configuring the Digital Unbalanced output connector settings from LabVIEW. It is similar in structure to the example shown in Figure 71.

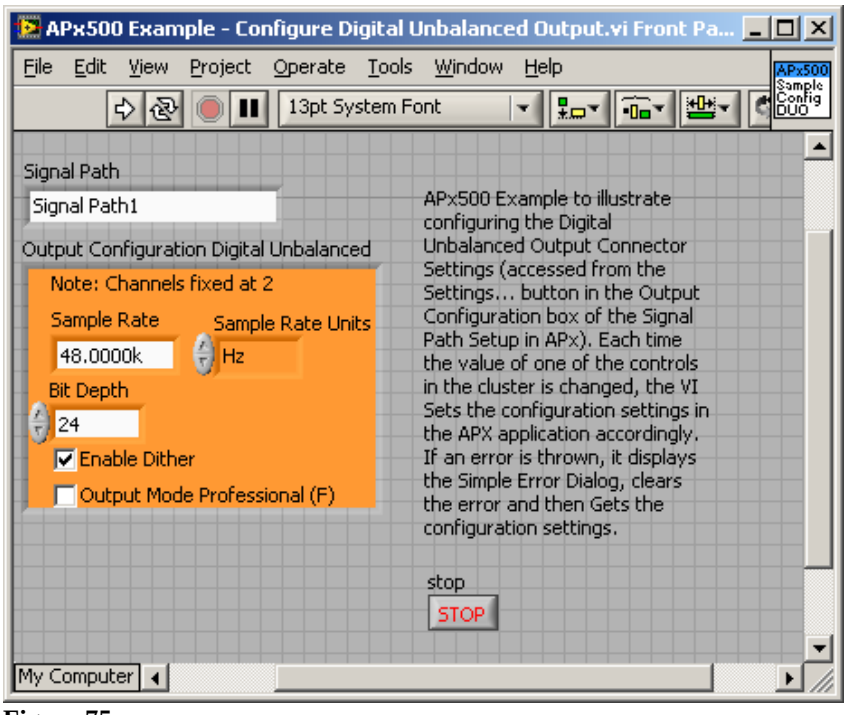


Figure 75

Reference Levels

There are two sets of VIs in the driver collection for dealing with APx500 reference levels. The first is the VI named *APx500 Utility-Get Set Reference Levels*. This VI is similar in function to the other configuration VIs. A cluster named Reference Levels is passed into and out of this VI. The context help for the VI and the Reference Levels cluster are shown in Figure 76. Note the similarity of the controls in the cluster to those on the References page (accessed from the References… button near the bottom of the APx500 screen in the Reference Levels measurement).

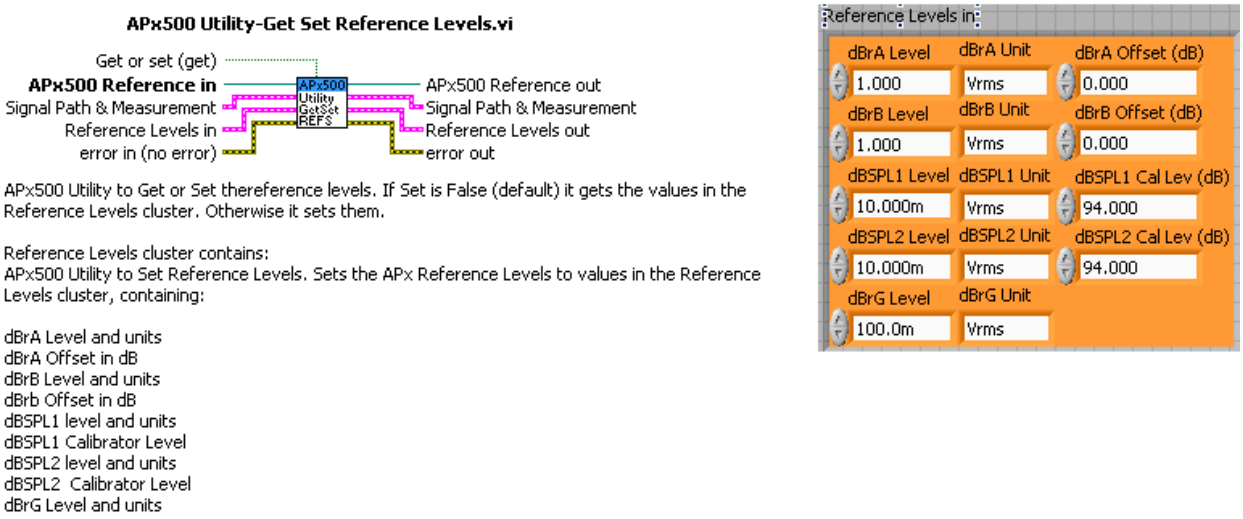


Figure 76

To set a reference level in LabVIEW, the procedure would be (1) get the reference levels cluster to determine the current settings, (2) use the Bundle by Name function to change the value of the reference level you want to change, and (3) set the reference levels using this VI to update APx500.

There are two additional VIs in the driver collection related to reference levels. These VIs are for accessing the auto-set generator level feature in APx500. The window in Figure 77 becomes visible in APx when you click the Auto Gen Level… button in the Signal Generation box of the Reference Levels measurement.

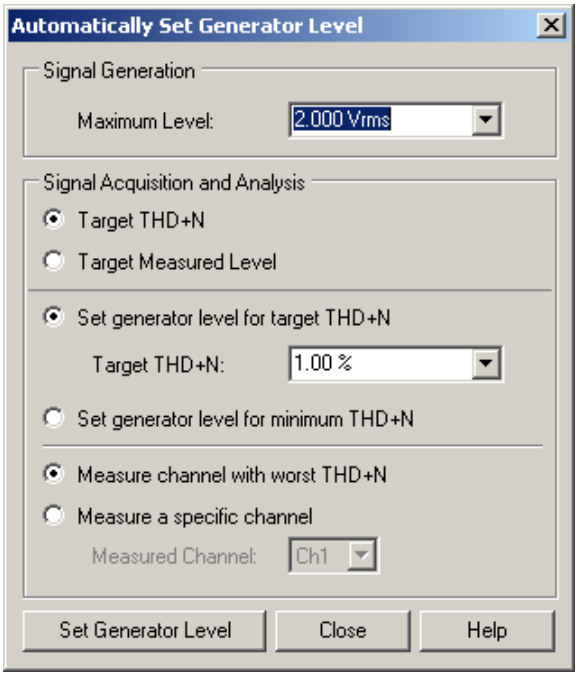


Figure 77

The VI on the left side of Figure 78, called *Reference Levels GetSet Auto Gen Level.vi* is used to get or set the auto-set generator level settings. The VI on the right side of Figure 78 is used to initiate the auto-set generator level measurement. It is equivalent to pressing the Set Generator Level button (at the bottom of the window shown in Figure 77.

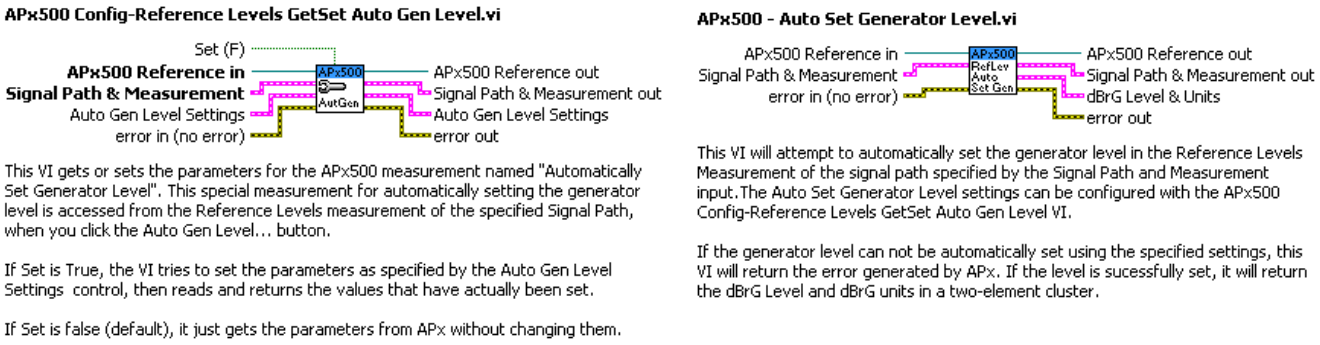


Figure 78

Note that the Auto Gen Level feature is a type of regulation measurement that iteratively tries to find a generator level that produces the specified distortion. If it can not find a suitable generator level, APx will generate an error that will be returned to LabVIEW. However, the error message passed to LabVIEW in this case is not as easy to understand as the one generated by APx (Figure 78).

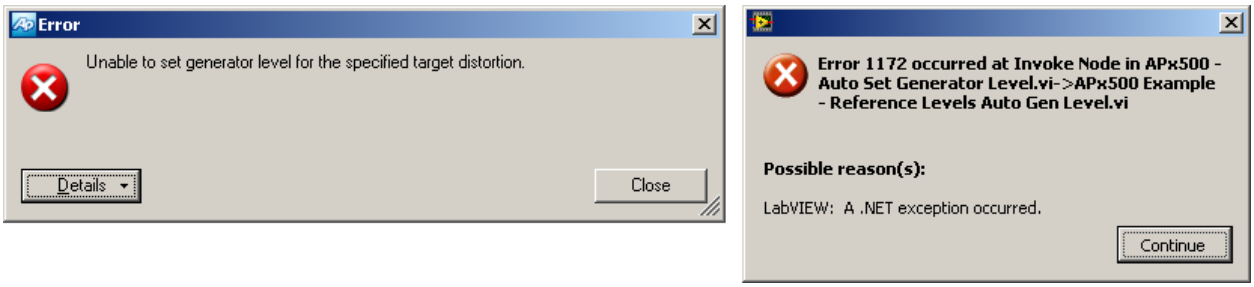


Figure 79

Figure 80 shows the front panel of an example VI included in the driver collection that illustrates working with auto-generator level VIs.

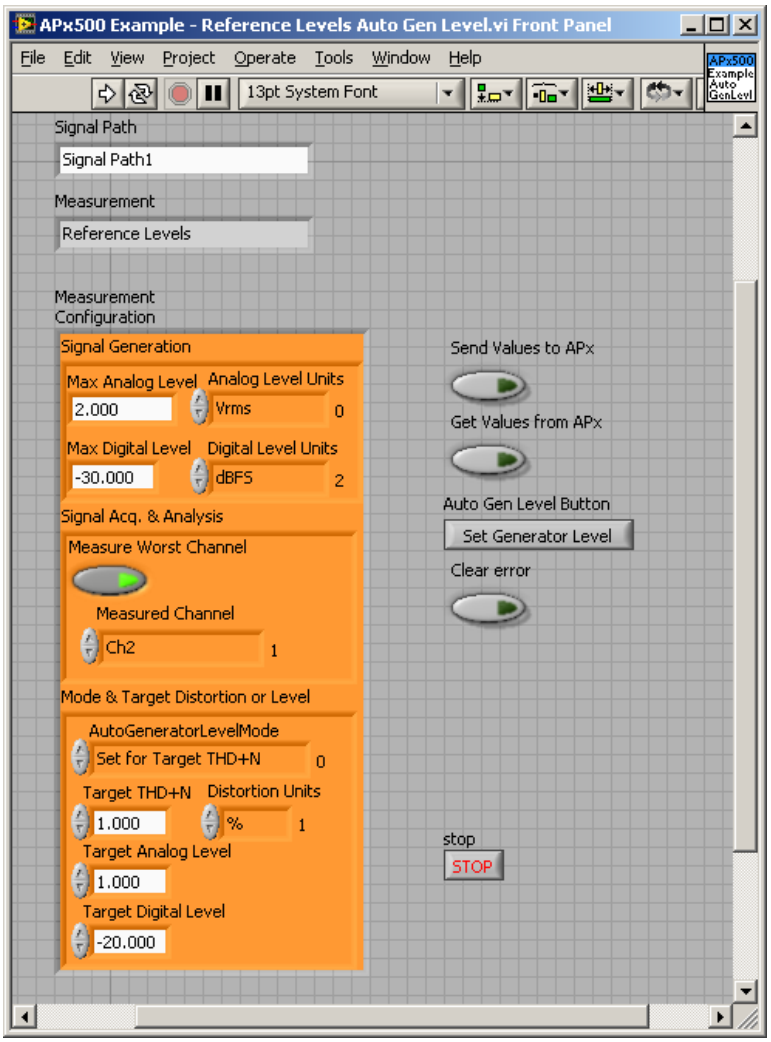


Figure 80

Acquiring Raw Data from the Signal Analyzer Measurement

The measurement VIs discussed so far contain data equivalent to what is displayed in the APx500 XY graphs and bar graphs. This is ideal for most applications. However, there may be situations in which a user wants to access *all* of the measurement data. Figure 81 shows two VIs

that are included for this purpose. The VI on the left runs the Signal Analyzer measurement and returns an array of LabVIEW waveform data containing all the acquired points. The VI on the right is similar, except that it returns an array of FFT spectra containing all FFT points. These VIs are useful if you want to perform FFT calculations, or conduct some analysis that is not included in the APx500 application.

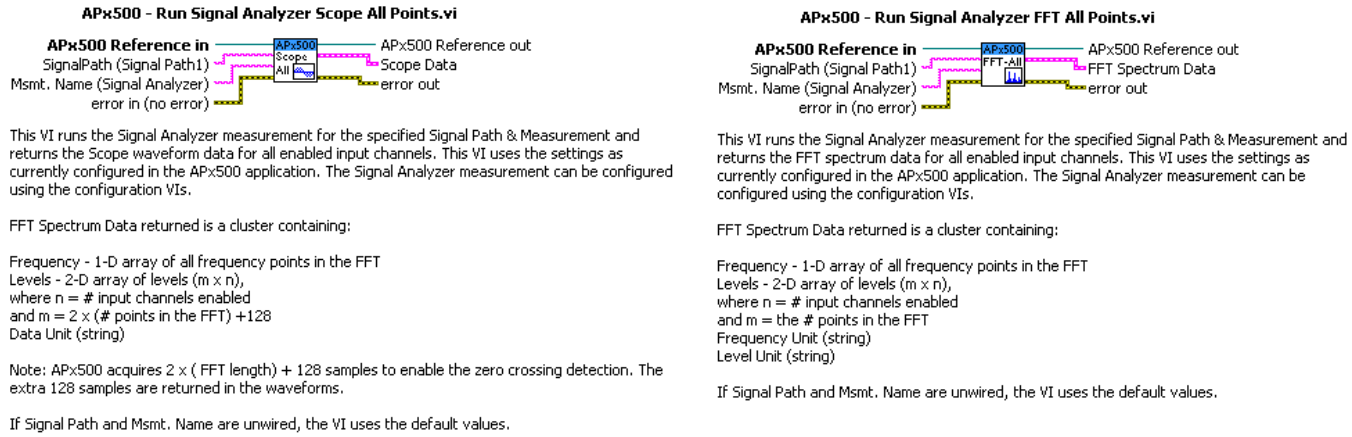


Figure 81

Controlling Input and Output Switchers

A set of four VIs to control switchers is included on the Switchers sub-palette (available from the Signal Path sub-palette of the Configuration palette) as shown in Figure 82. These can be used to get or set the input switcher configuration, get or set the output switcher configuration, open a switcher configuration file or save a switcher configuration file. The context help for the Get/Set Input Switcher VI is shown in Figure 83.

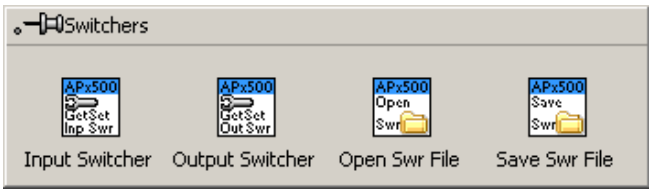


Figure 82

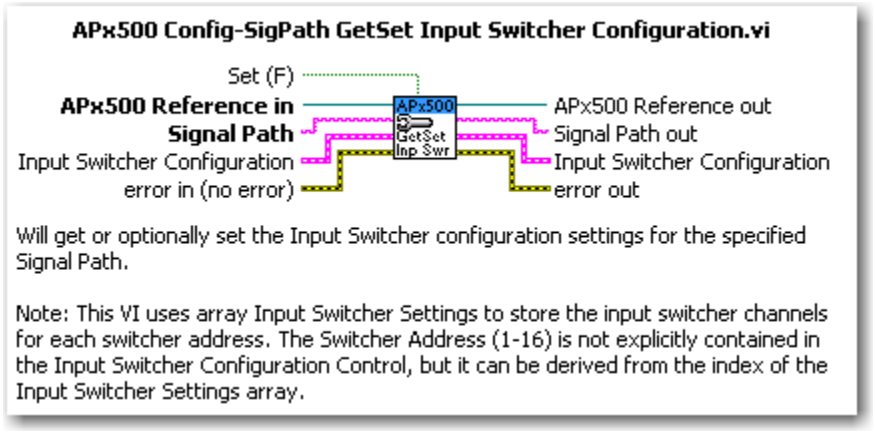


Figure 83

Using the WaveReader DLL

The WaveReader DLL provides functions that can be called from an external program, to enable transfer of acquisition data from an APx analyzer to that program in near real time. The DLL uses the APx500 Measurement Recorder measurement and its ability to save acquisition data to a .wav file. Two VIs that provide this functionality in LabVIEW are shipped with the driver (Figure 84 and Figure 85). These VIs are located with the DLL in the APx500 Examples LabVIEW project within the WaveReaderSupport folder (see Figure 12).

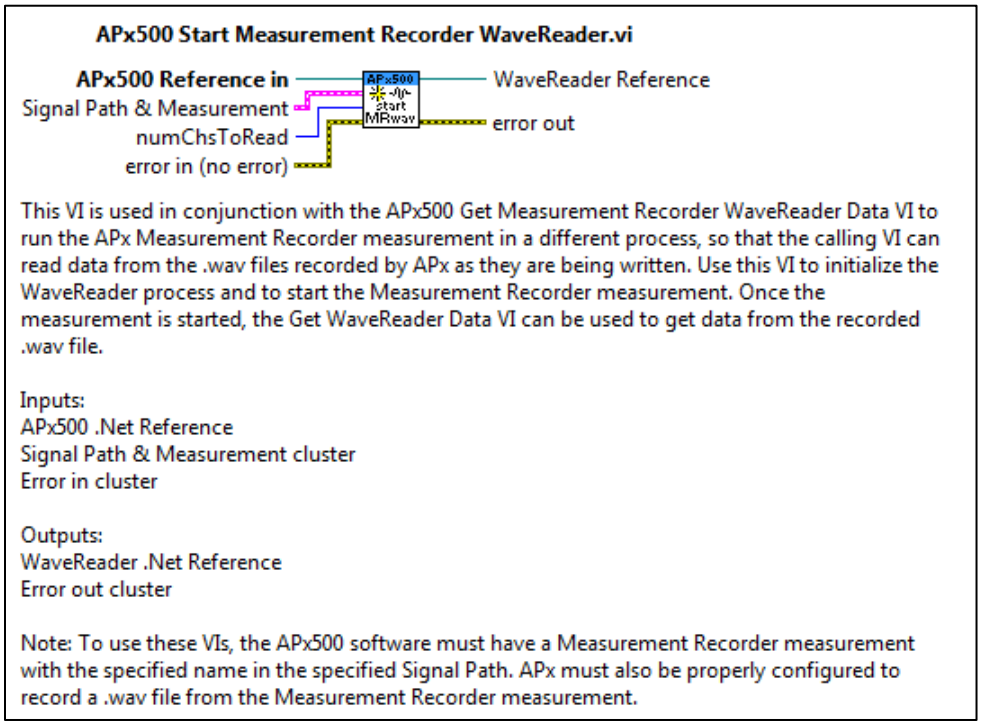


Figure 84

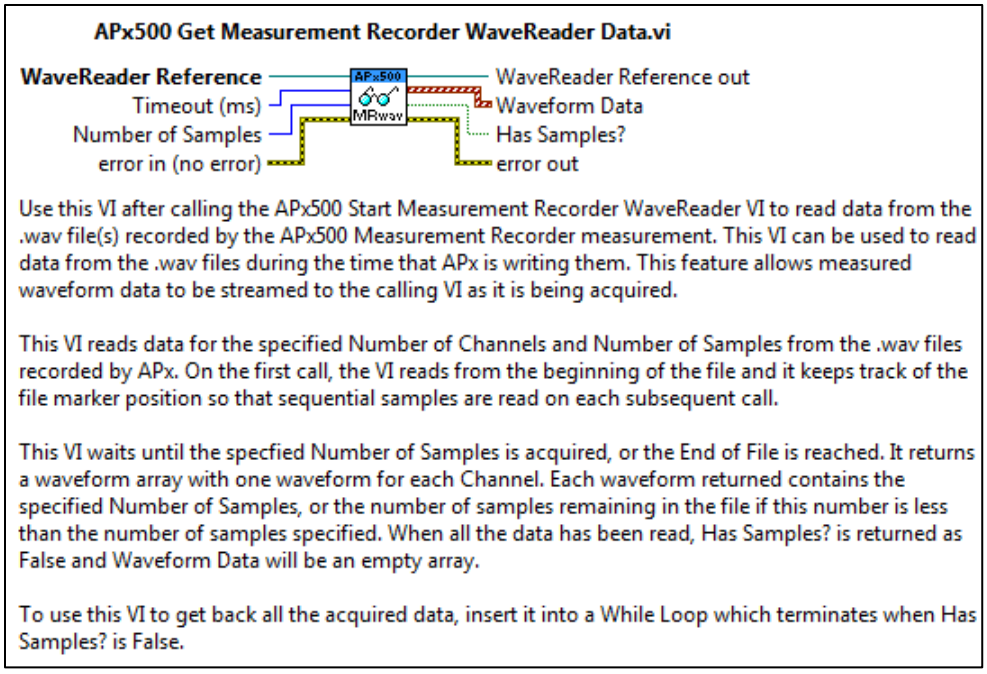


Figure 85

Figure 86 shows an example VI from the Driver Examples folder that illustrates using the WaveReader DLL. This example, with the accompanying APx project file (WaveReaderExample.approjx), illustrates acquiring two channels of data at a 48 kHz sample rate from APx500 with the VI's waveform and FFT graphs being updated every 0.1 seconds. The example uses a chirp signal that sweeps from 20 Hz to 20 kHz in 3 seconds.

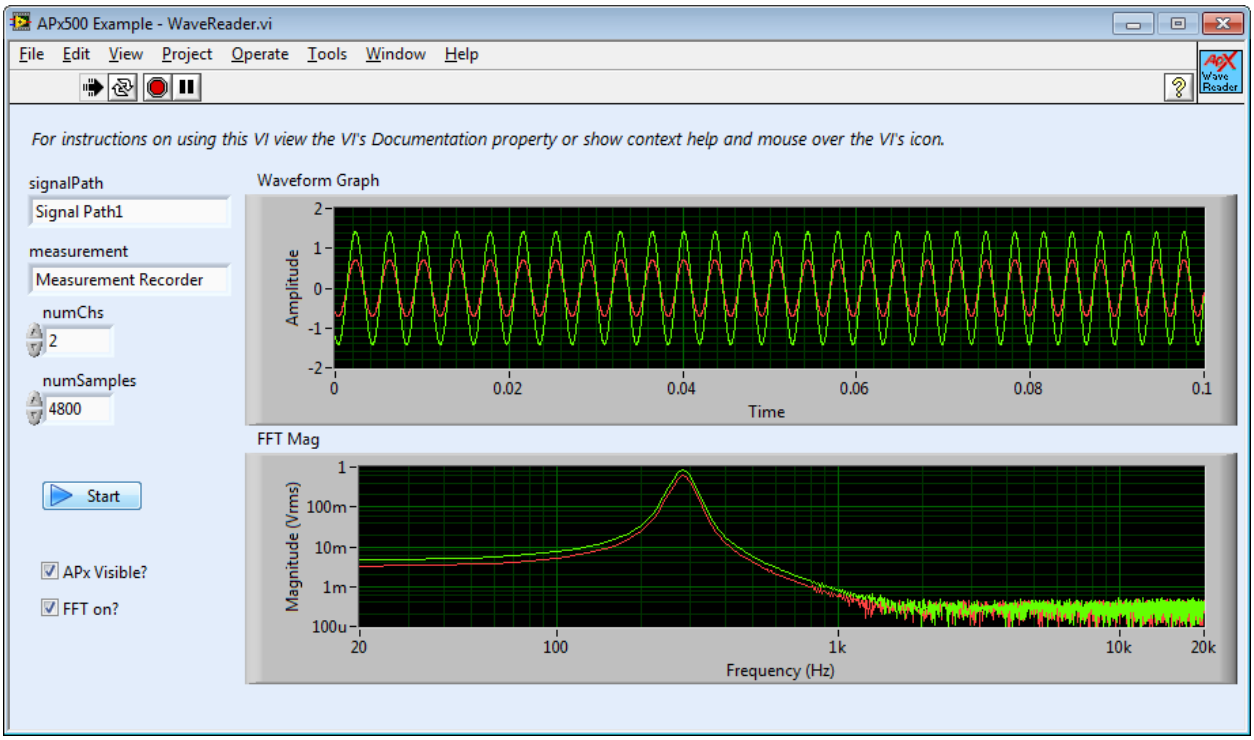


Figure 86

After loading the project file, be sure to change the Save to File directory in Measurement Recorder to a valid location on your PC. The Wavereader DLL reads a maximum of 64k samples but can be called in a loop, as shown in the example, to read any length. To make sure that memory is released, the LabVIEW .NET *Close Reference* VI should be used to close the Wavereader reference after use.

Directly Accessing .NET Methods and Properties

There is additional advanced functionality in APx500 that is not built into the provided drivers. However, you can access this functionality by modifying the existing VIs or by constructing your own VIs to access any method or property that exists in the APx500 .NET API. To see all the methods and properties available, open the API Browser from the APx500 folder in the Windows Start menu.

Example: Adding a Method

The following example will step through constructing a VI to invoke a method. This VI will independently set the generator level for each channel in the Level and Gain measurement. In the APx500 UI, this functionality is provided by clicking the Advanced Settings button in the Settings Panel of the Level and Gain measurement. In general, the settings and options in the Advanced Settings dialog of most measurements are not already provided in the LabVIEW driver.

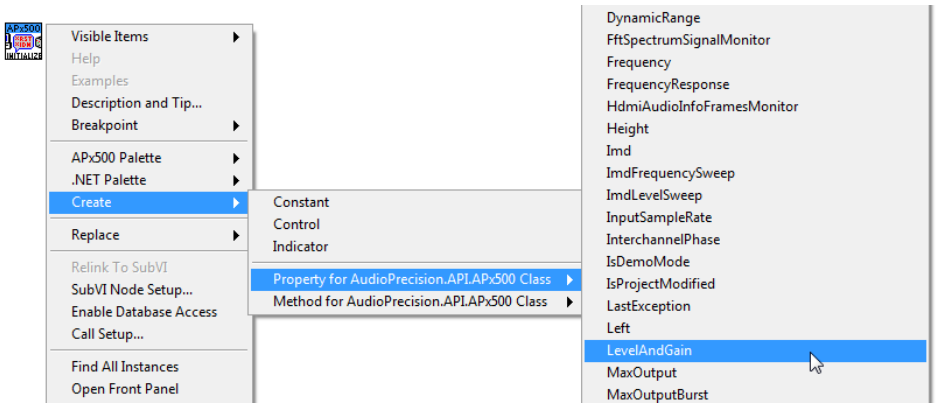


Figure 87

a) Place an APx Open VI. Then, right click on the APx500 Reference Out terminal of the VI, select Create > Property for AudioPrecision.API.APx500 Class, and choose LevelAndGain. Note that you can also perform this operation on the APx500 Reference Out terminal of any existing APx LabVIEW .NET Driver VI.

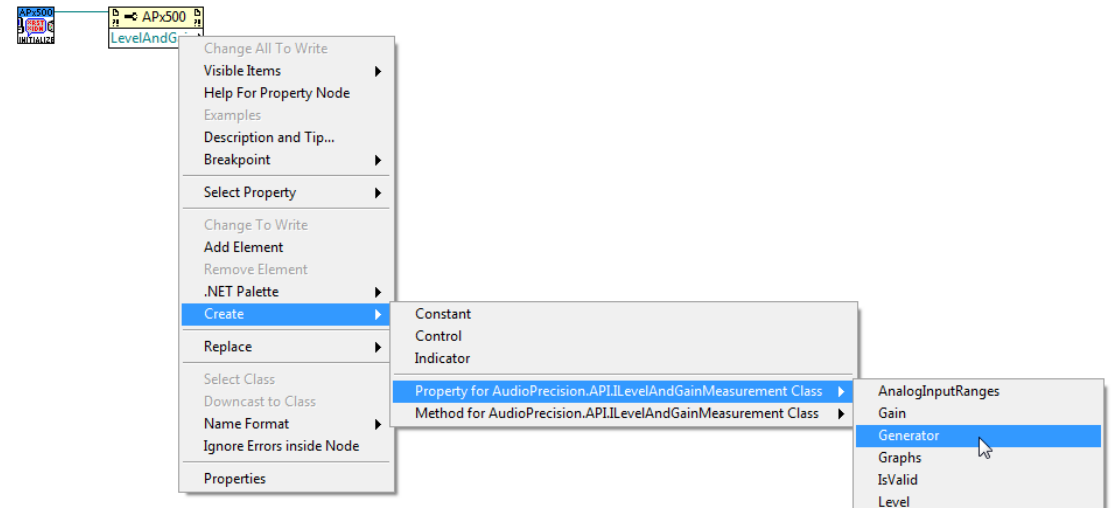


Figure 88

b) From the LevelAndGain property, right click and create a new Generator property for the ILevelAndGainMeasurement class. Wire it to the LevelAndGain property.

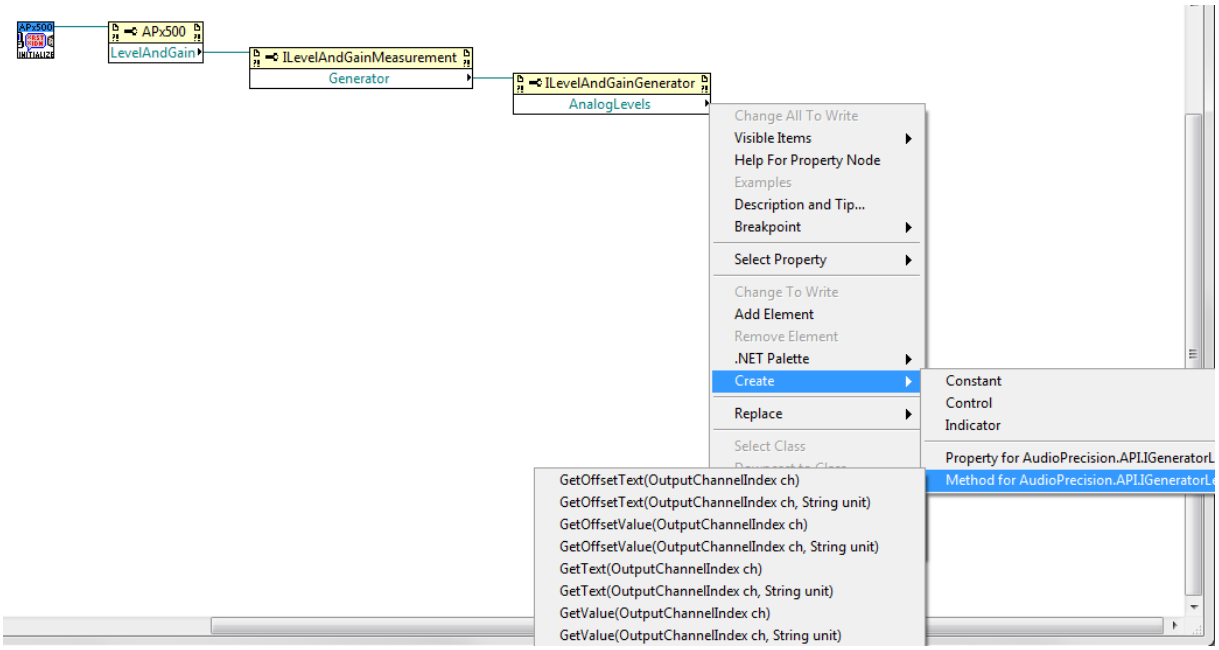


Figure 89

c) Connect the new Generator property, and then in a similar way add an AnalogLevels property to the ILevelAndGainGenerator class. From the AnalogLevels property, create a SetValue method and connect it.

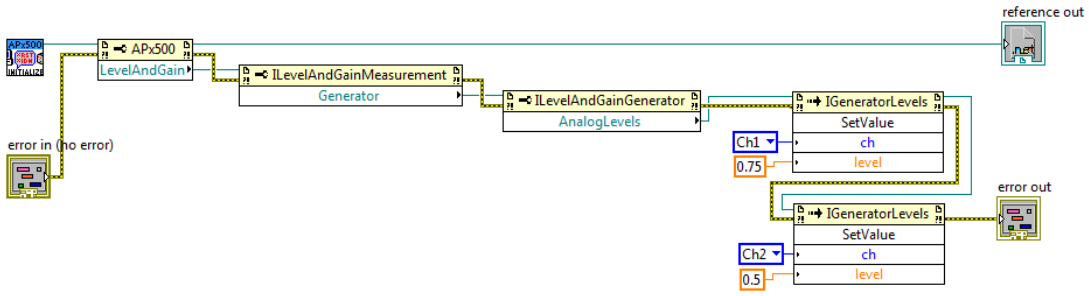


Figure 90

d) Add an additional SetValue method for channel 2 and complete the VI as shown above.

Example: Adding a Property

a) Place an APx Open VI. Then, right click on the APx500 Reference Out terminal of the VI, select Create > Property for AudioPrecision.API.APx500 Class, and choose LevelAndGain. Note that you can also perform this operation on the APx500 Reference Out terminal of any existing APx LabVIEW .NET Driver VI.

b) From the LevelAndGain property, right click and create a new Generator property for the ILevelAndGainMeasurement class. Wire it to the LevelAndGain property.

c) From the Generator property, right click and create a new On property for the ILevelAndGainMeasurement class. Wire it to the LevelAndGain property. Expand the property node and add an AnalogSineMode property.

d) Add indicators for the On and AnalogSineMode properties.

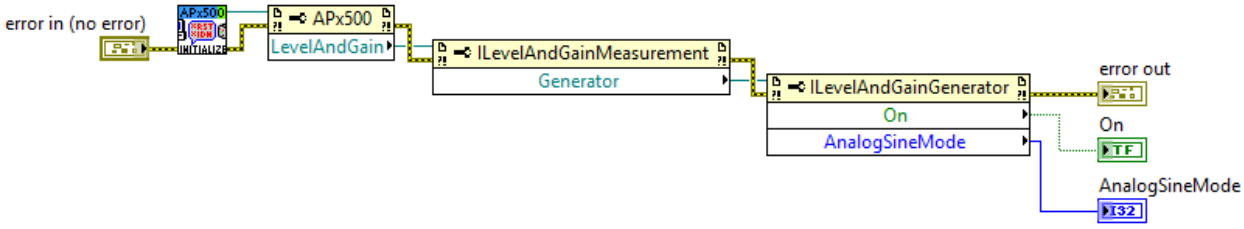


Figure 91

e) Many properties can set to either read or write. To change a property from write to read, right-click on the property and choose “Change To Write”.

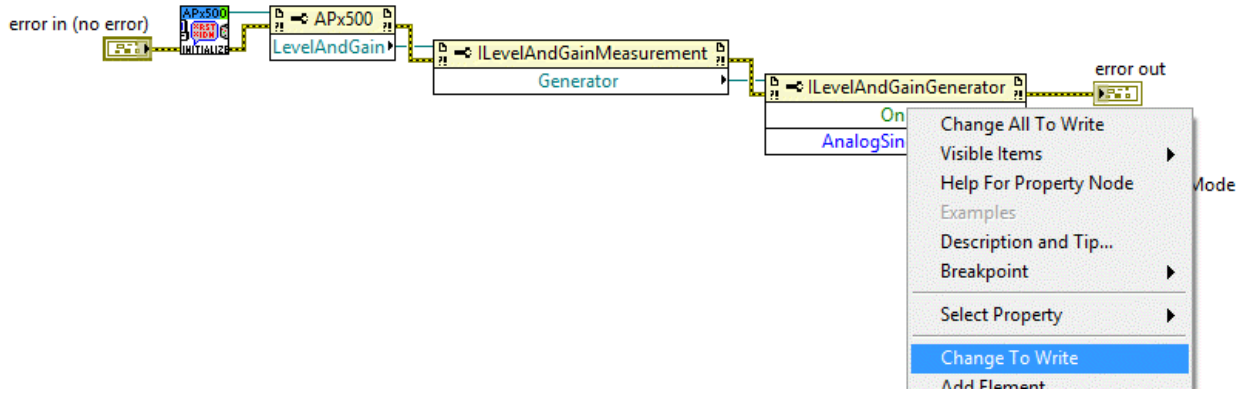


Figure 92

f) After changing the property to write, wire a control or a constant to it.

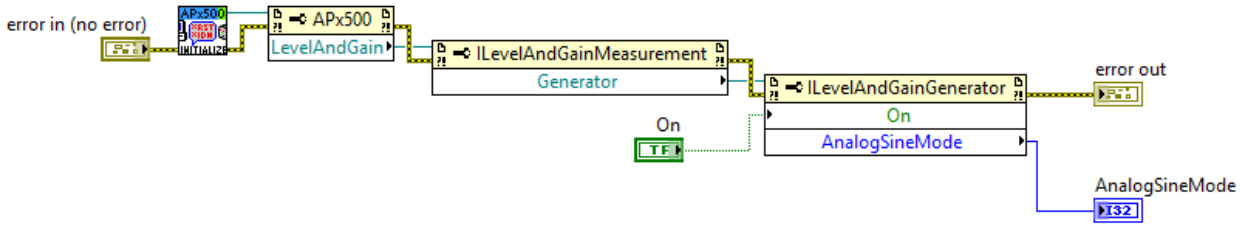


Figure 93

You can see additional examples by simply opening up any VI in the APx LabVIEW .NET Driver and opening its sub VIs until you see the .NET methods and properties. In this way, you can also modify any of the provided VIs to access to additional functionality. However, if you make modifications, be sure to save the VI with a new name so that your changes don't get overwritten the next time you upgrade the driver.

Conclusion

We hope that the APx LabVIEW .NET Driver will be a good resource for LabVIEW developers wanting to control and interact with an APx audio analyzer. If you need additional information, check the downloads and knowledge base sections at www.ap.com or call our Technical Support department for assistance.