Introduction to Programming the APx500 API

Contents

Introduction

The APx500 software provides a rich user interface which allows users to easily configure audio measurements. The measurement navigator and built in sequencing features allow users to quickly create an automated test sequence which creates a formatted report.

However, there are many cases where programmatic control is required. Test systems often integrate more than one device, such as power supplies, load resistors, etc. Other systems may require more detailed data extraction and reporting.

To meet these goals, the APx500 software offers a rich programming interface which encompasses nearly all features found in the APx500 user interface. This programming interface is defined using Microsoft .NET technologies, and as such, can be accessed using a variety of programming tools including Microsoft VB.NET, Microsoft C#, and National Instruments LabView. Other tools are supported as long as they support some sort of .NET interface.

The APx500 software differs from legacy Audio Precision software such as ATS-2 and AP2700 in that it does not use COM or ActiveX and it does not have a built in code editor or macro recorder. However, free "Express" programming tools from Microsoft can easily provide a rich and powerful programming environment.

This document will introduce the API, describing how it relates to the user interface, and how data can be extracted from the APx500 software. More code examples can be found at www.ap.com.

A quick tour of the APx500 user interface

The APx500 user interface allows users to easily create project files with one or more signal paths and many measurements. Project files can be saved and re-used. These saved project files can easily be loaded and run via the API.

Sequence Mode

Sequence Mode provides users with a set of measurements in which all applicable measurement parameters are grouped together to provide a specific set of results. Sequence Mode can automatically run one or more measurements sequentially, optionally producing a report at the end of the run.

Signal Paths

A signal path describes the physical input and output interfaces used to communicate between the APx500 instrument and the device under test. The signal path settings are controlled via the Signal Path Setup measurement.

Measurements

APx500 has many built in measurements which encapsulate both signal generation and signal analysis. Some measurements take instantaneous readings, while others have algorithms to measure audio parameters while modulating frequency or signal level.

Results

Each measurement produces one or more graphs or results which are computed based on the algorithm and settings for the measurement. There are many types of results including: meters (bar graphs), XY graphs, and vertical bar graphs.

Navigator

The measurement navigator contains all of the signal paths, measurements, and results which are in the APx500 project file.

Signal paths, measurements, and results can all be checked or un-checked to indicate whether or not they are part of the current sequence.

The sequence contains a report which can be configured to use a built in report style, or a custom report layout using a Microsoft Word document.

Measurements and results can be added and removed from the project. Additional results can be derived from the default results to perform statistical analysis, or other calculations such as normalization or smoothing.

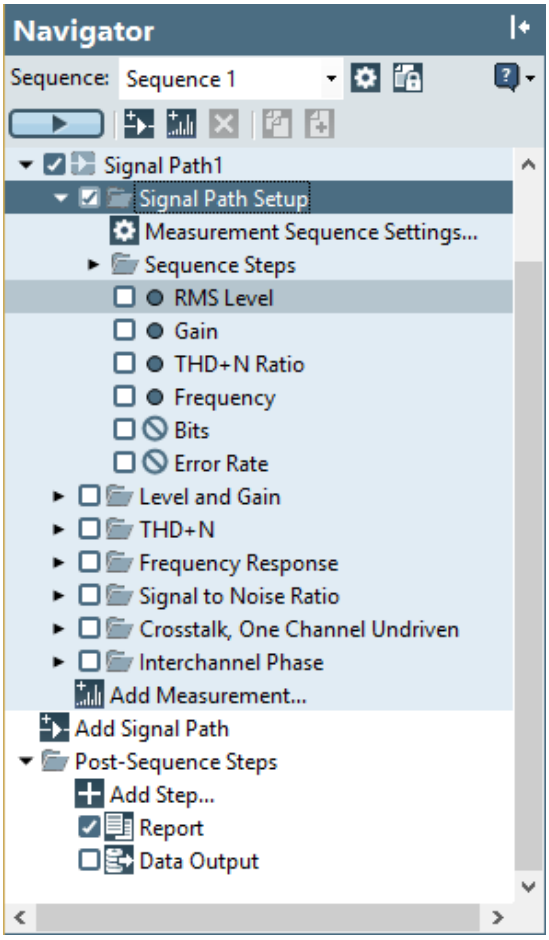


Figure 1 - Navigator

Bench Mode

Bench Mode provides users with a more flexible configuration for making measurements. Generator settings are independent from analysis settings. Measurement results can be selected by the user. No sequence automation is available in Bench Mode.

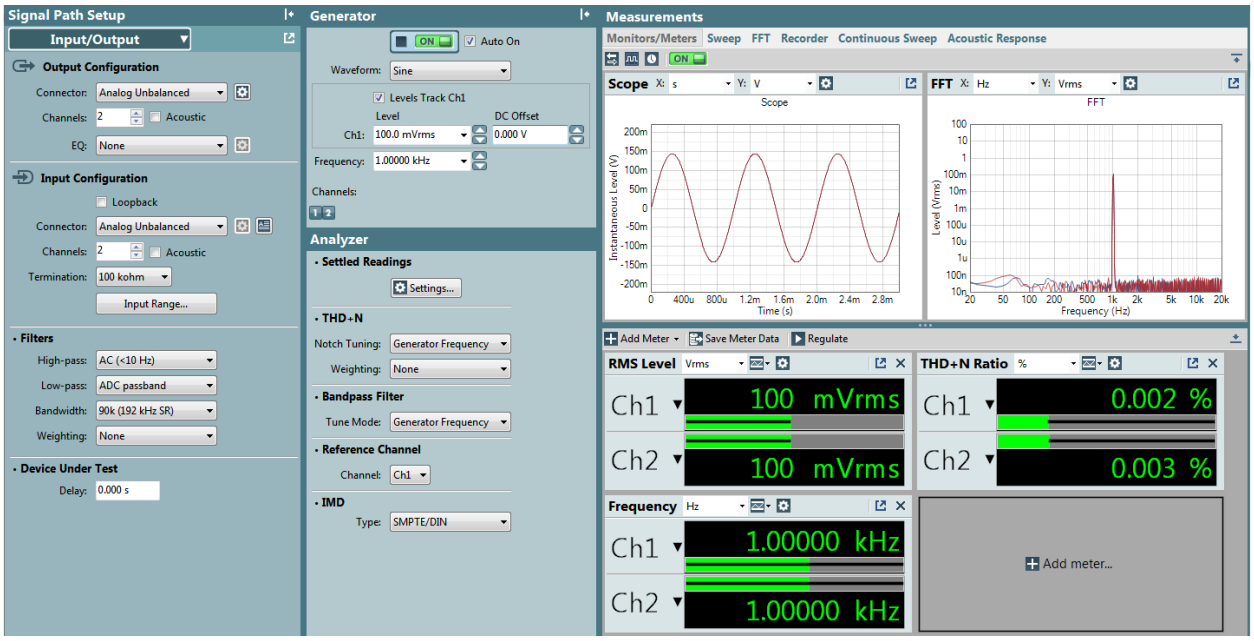


Figure 2 - Bench Mode

Signal Path Setup

Like Sequence Mode, Bench Mode has Signal Path Setup parameters which describe the physical input and output interfaces used to communicate between the APx500 instrument and the device under test.

Generator/Analysis Settings

Bench Mode has independent settings for signal generator and analysis.

Meters

Bench Mode provides a configurable set of instantaneous meter readings. Settled reading values can be obtained using the APx API.

Measurements

Bench Mode offers a fixed selection of measurements which provide both X,Y graphs and meter results. The Sweep measurement has configuration parameters which control which attributes of APx will be modified and which results will be measured. The Recorder measurement provides readings as a function of time.

APx500 API structure

The APx500 API is written using Microsoft .NET language tools. In .NET, DLL projects are called "assemblies". The APx500 API is an assembly. Each assembly has a version number. For each APx500 software version, there is a matching API assembly with the same version number. To access code in an assembly, you must add a "reference" to that assembly. Adding references will be covered in the language specific sections of this document.

The APx500 API is object oriented. All API calls start with the root object of the API which is of type `APx500`. From the root API object, the user has access to all of the signal paths, measurements, results, and other settings available in the APx500 software. The `OperatingMode` setting determines which part of the APx software is active. Only one mode can be programmed at a time, otherwise exceptions are thrown from APx.

The API follows .NET programming patterns using objects called collections to represent multiple items grouped together. The API uses an interface called `IEnumerable` to allow easy iteration over the contents of a collection. Languages such as VB.NET have a construct called a `For`..`Each` loop to iterate over a collection. In C#, the language construct is called `foreach`.

VB.NET	C#

`For Each SP As ISignalPath In APx.Sequence`
 	
`Next`
	`foreach (ISignalPath sp in APx.Sequence)`
	`{`
	`}`

Whenever possible, the API uses strongly typed objects so casting objects is very rarely required. Many properties and methods in the API use enumerations to specify a limited set of values that are applicable to a particular setting. Microsoft development tools like Visual Studio (and Express) offer code completion hints when typing code.

API Tips

When writing programs using the APx API, a few items are useful to keep in mind:

- When setting a measurement parameter such as generator level or generator frequency, always set the unit before setting the value
- Always give signal paths and measurements unique names so you can properly access them via the API
- Always activate a measurement before accessing its settings by calling `APx.ShowMeasurement`
- Many API calls can throw exceptions if an error occurs. Use the Try..Catch language syntax to handle errors

APIBrowser

Audio Precision provides a tool to help users navigate the API. The APIBrowser is available in the Windows Start menu under the Audio Precision <version>\API folder.

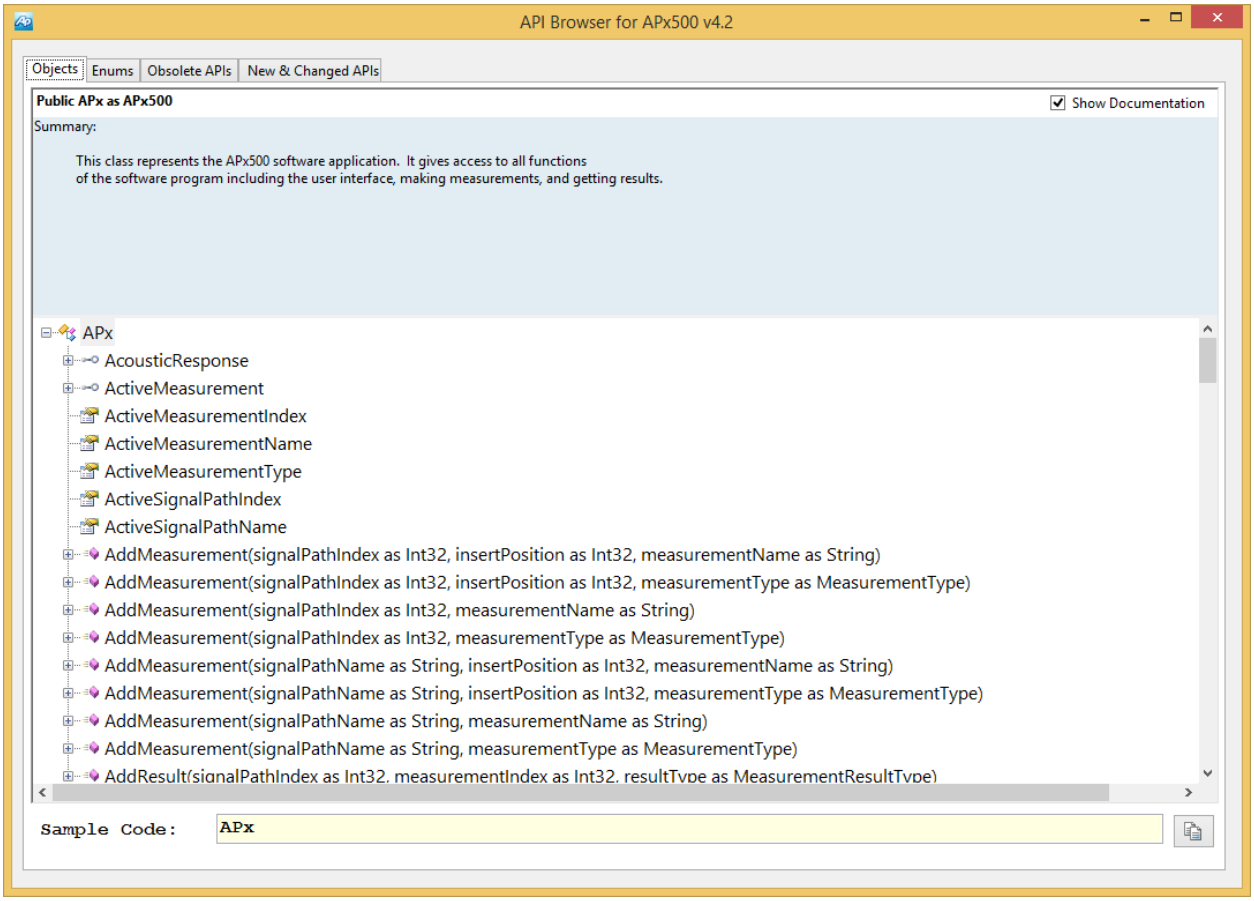


Figure 3 – APIBrowser

API Help File

The APx500 application also comes with a searchable help file in Microsoft Help format. The file is called APx500_API_PRG.chm and it is available in the Windows Start menu under the Audio Precision <version>\API folder.

Creating projects in VB.NET

Microsoft provides a free "Express" version of VB.NET from their website: www.microsoft.com/express. You can also use the full Microsoft Visual Studio tools if you require more features.

Audio Precision provides project templates to make getting starting with VB.NET easy. These templates can be downloaded from www.ap.com/download. After the templates are installed, create a new project and choose "APx500 Application".

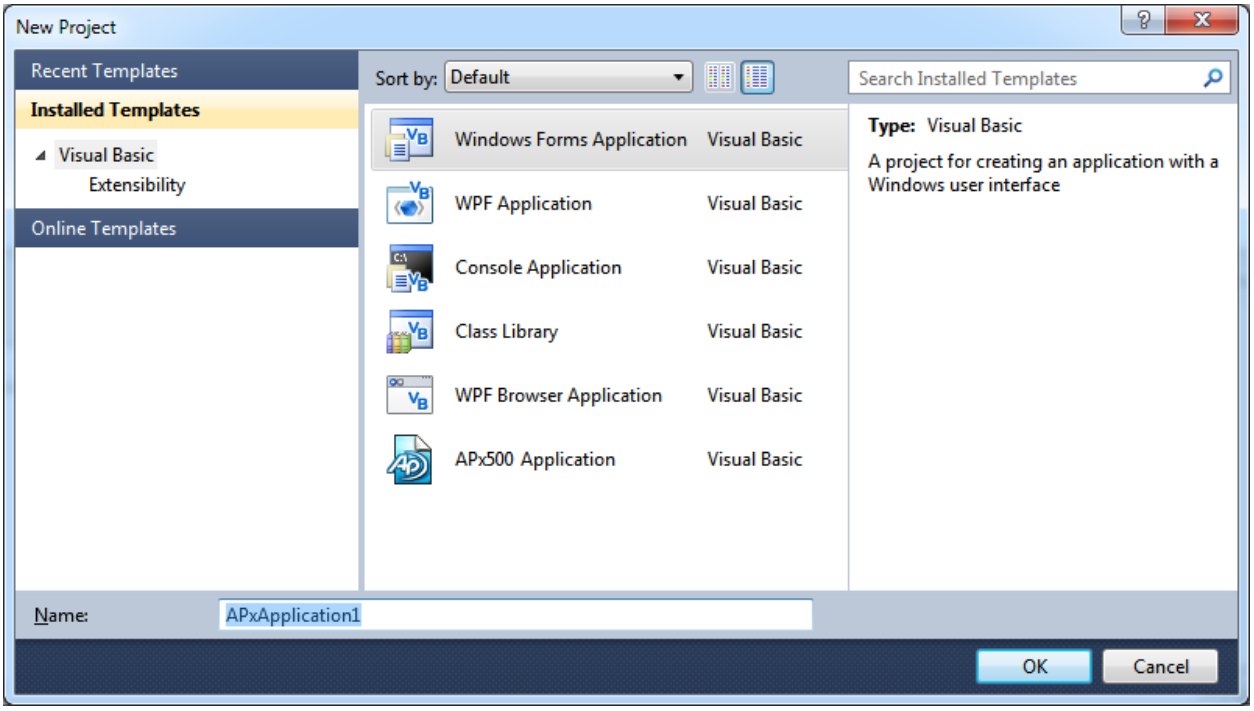


Figure 4 - Create New Project

If you do not use a project template, you must add a reference to the API assembly before you can make API calls. To add a reference, choose the Project | Properties menu item.

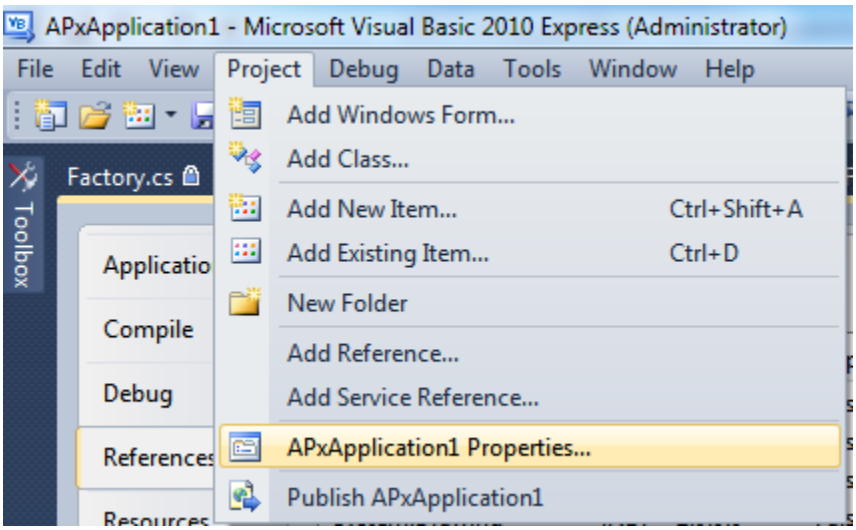


Figure 5 - Edit Project Properties

On the “References” tab, choose the “Add…” button.

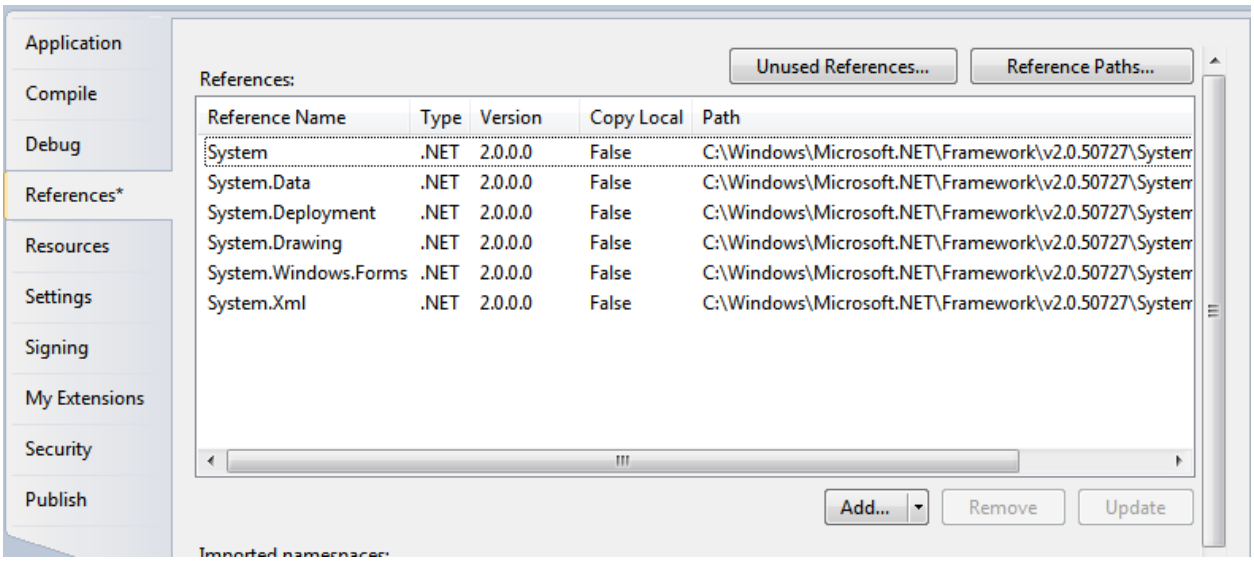


Figure 6 – Project References

In the Add Reference dialog, select the ".NET" tab and find the AudioPrecision.API version you are interested in using.

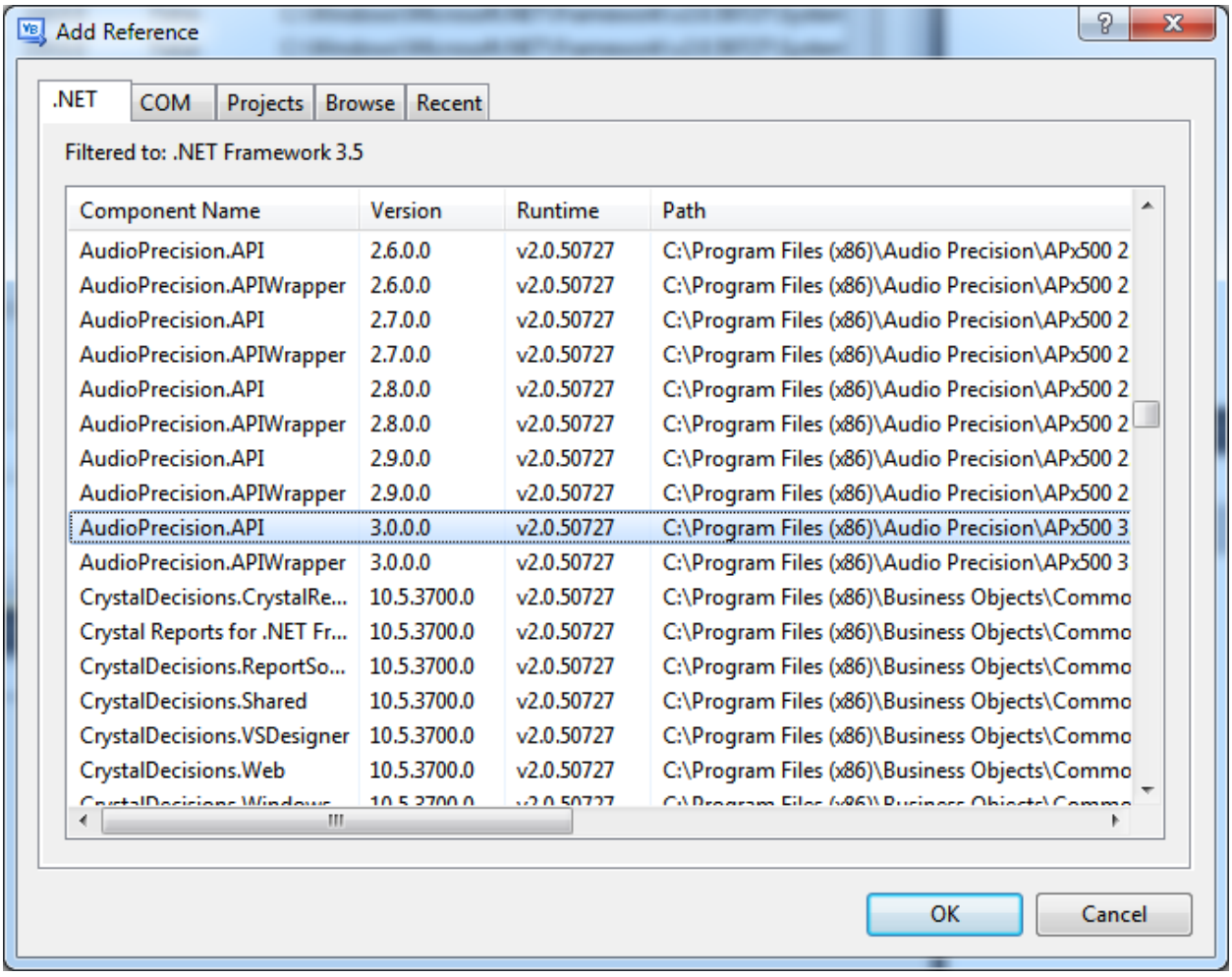


Figure 7 - Add Reference

Once you have selected the AudioPrecision.API assembly, you will need to import the AudioPrecision.API namespace.

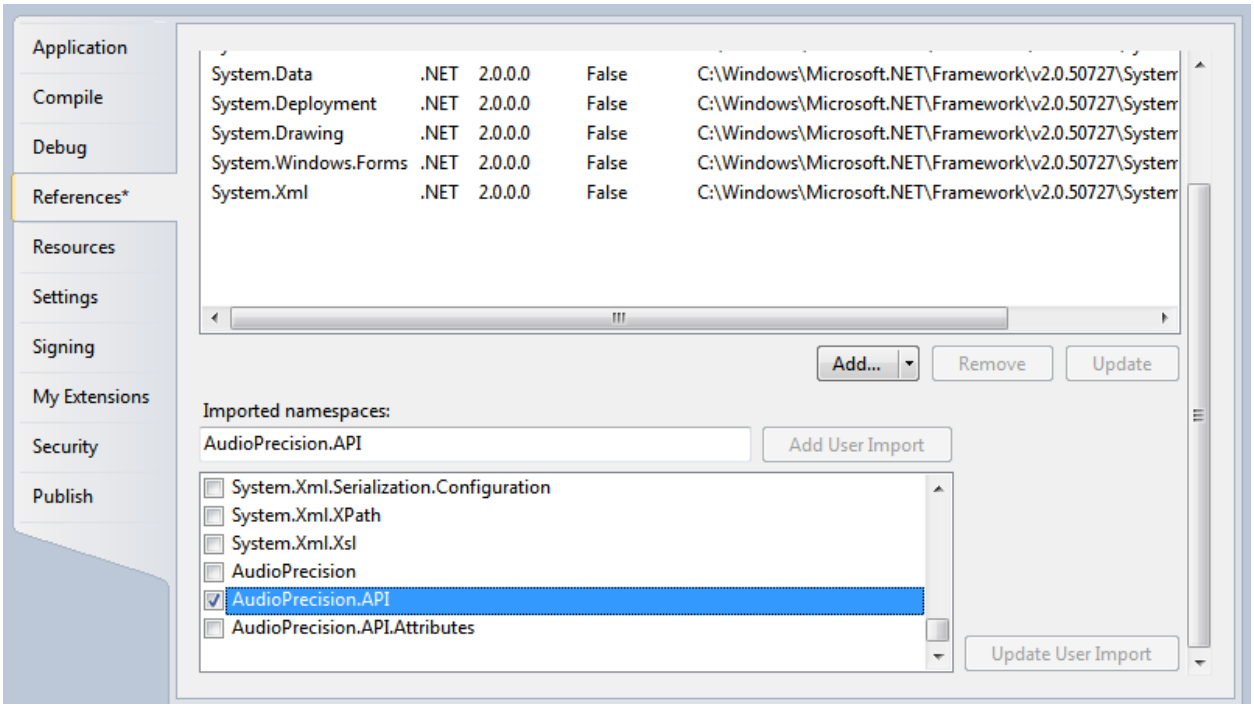


Figure 8 - Import Namespace

You are now ready to write an APx500 API program.

Creating projects in C#

Microsoft provides a free “Express” version of C# from their website: www.microsoft.com/express. You can also use the full Microsoft Visual Studio tools if you require more features.

Audio Precision provides project templates to make getting starting with C# easy. These templates can be downloaded from www.ap.com/download. After the templates are installed, create a new project and choose “APx500 Application”.

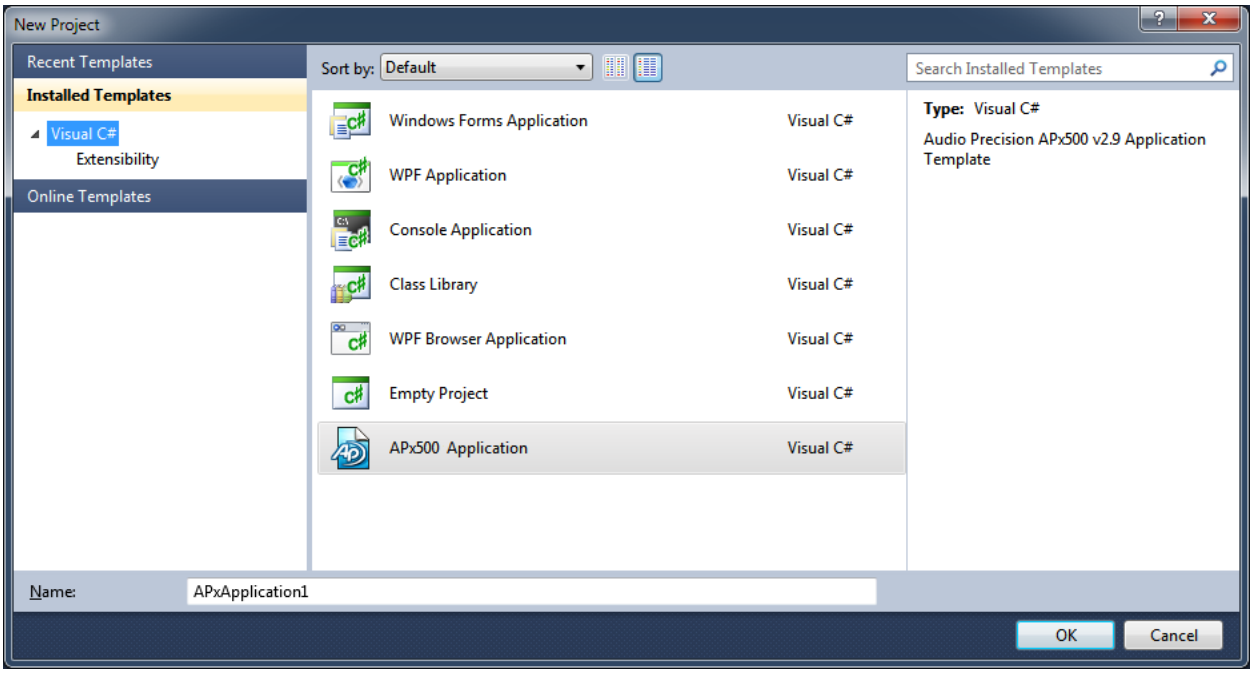


Figure 8 - Create New Project

If you do not use a project template, you must add a reference to the API assembly before you can make API calls. To add a reference, right click on the "References" folder in the Solution Explorer window and choose "Add Reference…"

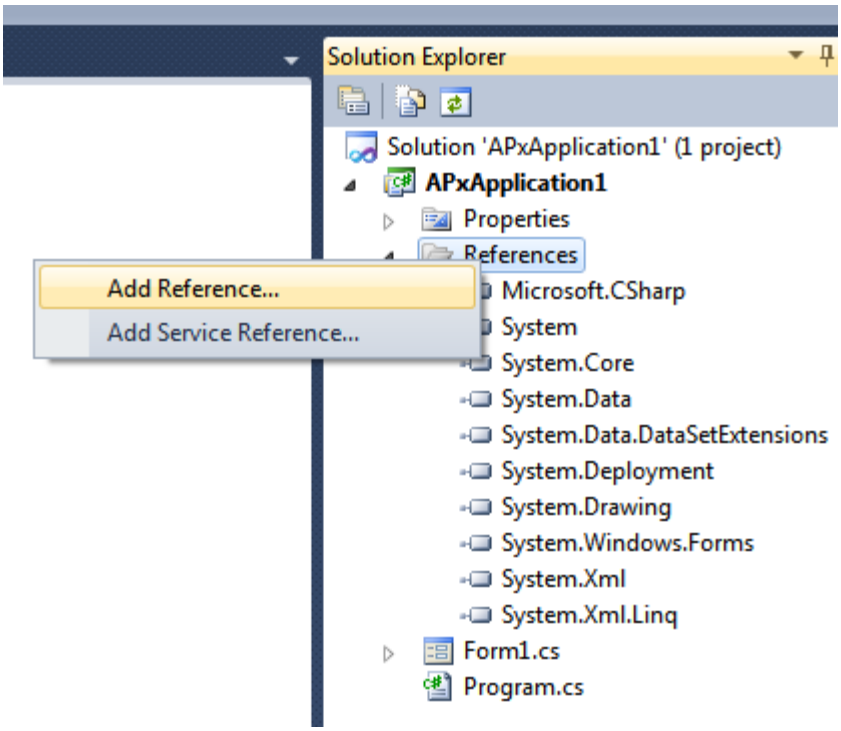


Figure 9 – Add Reference to Solution Explorer

In the Add Reference dialog, select the ".NET" tab and find the AudioPrecision.API version you are interested in using.

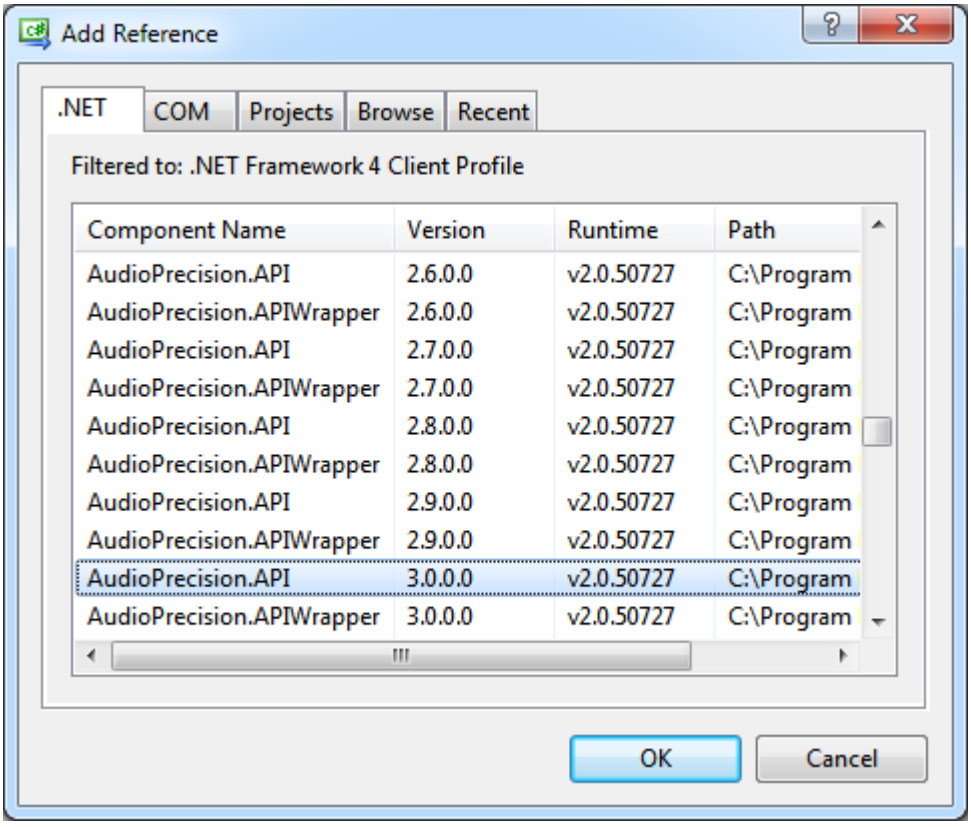


Figure 10 - Add Reference

You are now ready to write an APx500 API program.

Creating projects in LabView

Audio Precision provides an instrument driver for LabView users. The driver manages the APx500 automation interface and provides VIs which can be used in your LabView program. For more information, see the website: http://ap.com/products/apx/labview.

First code examples – starting the application and opening a project file

Once you have created your project, you are ready to start using the API to automate tasks in APx500. Code examples will be given in VB.NET and C#. More examples are available from the Audio Precision website: www.ap.com.

All API calls start with the `APx500` object constructor. This call connects your program to the APx software. The APx500 object has many properties. One property is called `Visible` and it controls whether or not the application window can be seen by the user.

Start the application and make it visible (VB.NET)

```
'Declare APx variable and connect to the APx software
Dim APx As New APx500

'Show the APx500 application window
APx.Visible = True

'Open a project file
APx.OpenProject("C:\Project.approjx")
```

Start the application and make it visible (C#)

```
//Declare APx variable and connect to the APx software
APx500 APx = new APx500();

//Show the APx500 application window
APx.Visible = true;

//Open a project file
APx.OpenProject("C:\\Project.approjx");
```

Selecting the operating Mode (VB.NET)

```
'Start APx in Bench Mode
Dim APx As New APx500(APxOperatingMode.BenchMode)
APx.Visible = True
```

```
'Start APx in Sequence Mode
Dim APx As New APx500(APxOperatingMode.SequenceMode)
APx.Visible = True
```

```
'Change to Bench Mode
APx.OperatingMode = APxOperatingMode.BenchMode
```

```
'Change to Sequence Mode
APx.OperatingMode = APxOperatingMode.SequenceMode
```

Selecting the operating Mode (C#)

```
//Start APx in Bench Mode
APx500 APx = new APx500(APxOperatingMode.BenchMode);
APx.Visible = true;
```

```
//Start APx in Sequence Mode
APx500 APx = new APx500(APxOperatingMode.SequenceMode);
APx.Visible = true;
```

```
//Change to Bench Mode
APx.OperatingMode = APxOperatingMode.BenchMode;
```

```
//Change to Sequence Mode
APx.OperatingMode = APxOperatingMode.SequenceMode;
```

Code examples - controlling system parameters

The following sections will describe how to control many of the various settings available in the APx500 user interface and API. Whenever possible, the API offers the same ability to control settings that is available in the application user interface.

The APx500 API is strongly typed. This means that each measurement has its own defined properties and methods encapsulated in an interface. The API is only allowed to make changes to the active measurement; therefore it is required to make the desired measurement visible before accessing its settings.

Controlling signal path settings (Sequence Mode)

It is often useful to configure the input or output connector and the settings associated with the selected connector. The following example demonstrates setting the input connector to "Analog Balanced", and configuring the some of the input settings.

VB.NET

```
'Set the input connector to Analog Balanced
APx.SignalPathSetup.InputConnector.Type = InputConnectorType.AnalogBalanced

'Set the number of input channels to 2
APx.SignalPathSetup.AnalogInputChannelCount = 2

'Set the analog ADC bandwidth
APx.SignalPathSetup.LowpassFilterAnalog = LowpassFilterModeAnalog.AdcPassband
APx.SignalPathSetup.LowpassFilterAnalogBandwidth = AdcBandwidth.bw90k

'Set the input termination to 600 ohm (for 52x instruments)
APx.SignalPathSetup.AnalogInput.SetTermination(InputChannelIndex.Ch1,
                              AnalogInputTermination.InputTermination_600)
```

C#

```
//Set the input connector to Analog Balanced
APx.SignalPathSetup.InputConnector.Type = InputConnectorType.AnalogBalanced;

//'Set the number of input channels to 2
APx.SignalPathSetup.AnalogInputChannelCount = 2;

//Set the analog ADC bandwidth
APx.SignalPathSetup.LowpassFilterAnalog = LowpassFilterModeAnalog.AdcPassband;
APx.SignalPathSetup.LowpassFilterAnalogBandwidth = AdcBandwidth.bw90k;

//Set the input termination to 600 ohm (for 52x instruments)
APx.SignalPathSetup.AnalogInput.SetTermination(InputChannelIndex.Ch1,
                               AnalogInputTermination.InputTermination_600);
```

Controlling generator settings (Sequence Mode)

Each measurement in an APx project has its own generator settings (except for measurements which do not have a generator). For measurements which have a generator, the settings are available via the `.Generator` property from the measurement.

All of the settings in the user interface generator panels are available via the API. This means settings such as Frequency, Level, and Waveform can all be configured.

Setting generator level and frequency

The following example configures the generator for THD+N measurement.

VB.NET

```
'Set the first channel of the analog generator to 100.0 mVrms
APx.ThdN.Generator.Levels.SetValue(OutputChannelIndex.Ch1, 0.1)

'Set the generator frequency to 2.000 kHz
APx.ThdN.Generator.Frequency.Value = 2000
```

C#

```
'Set the first channel of the analog generator to 100.0 mVrms
APx.ThdN.Generator.Levels.SetValue(OutputChannelIndex.Ch1, 0.1);

'Set the generator frequency to 2.000 kHz
APx.ThdN.Generator.Frequency.Value = 2000;
```

Selecting a different generator type

Some measurement generators support different waveform types such as “Sine”, “Square”, “Sine, Dual”, and “Sine, Var Phase”. The generator type can be accessed via the `.Waveform` property. The available generator types for a measurement can be enumerated using the `.WaveformNames` property.

The following example shows setting the Level and Gain measurement generator to “Sine, Dual”. If the specified waveform is spelled incorrectly, or is not supported, APx will throw an exception indicating the waveform selection is not valid. The selection is not case-sensitive.

VB.NET

```
'Set the generator waveform to Sine, Dual
APx.LevelAndGain.Generator.Waveform = "Sine, Dual"

'Set the Frequency A value
APx.LevelAndGain.Generator.Frequency.Value = 1000.0

'Set the Frequency B value
APx.LevelAndGain.Generator.SplitFrequency.FrequencyB.Value = 4000.0
```

C#

```
//Set the generator waveform to Sine, Dual
APx.LevelAndGain.Generator.Waveform = "Sine, Dual";
```

```
//Set the Frequency A value
APx.LevelAndGain.Generator.Frequency.Value = 1000.0;

//Set the Frequency B value
APx.LevelAndGain.Generator.SplitFrequency.FrequencyB.Value = 4000.0;
```

Loading and selecting a waveform

Many generators in APx can play a waveform from a file. Supported linear waveform types include .WAV, .AGM, .AGS. APx can also play encoded audio waveforms when the output connector is digital. Supported waveforms include many Dolby® and DTS® file formats.

Once a waveform file has been loaded into a generator, it is available for use by all other measurements in the project. The list of available waveform files for any measurement is accessed via the `.WaveformNames` property of the generator, or via the `APx.AttachedProjectItems API`.

VB.NET

```
'Select and load a file called Test1.wav for playback
APx.LevelAndGain.Generator.LoadWaveformFile("c:\WAV\Test1.wav")

'Select and load a file called Test2.wav for playback
APx.LevelAndGain.Generator.LoadWaveformFile("c:\WAV\Test2.wav")

'Re-select Test1.wav
APx.LevelAndGain.Generator.Waveform = "Test1.wav"
```

C#

```
//Select and load a file called Test1.wav for playback
APx.LevelAndGain.Generator.LoadWaveformFile(@"c:\WAV\Test1.wav");

//Select and load a file called Test2.wav for playback
APx.LevelAndGain.Generator.LoadWaveformFile(@"c:\WAV\Test2.wav");

//Re-select Test1.wav
APx.LevelAndGain.Generator.Waveform = "Test1.wav";
```

Controlling sweep settings

Measurements like Stepped Frequency Sweep and Frequency Response have generators with additional parameters such as start and stop frequency, and number of sweep points. All of these settings are available via the API.

VB.NET

```
'Set the start frequency to 20 Hz
APx.SteppedFrequencySweep.Generator.StartFrequency.Value = 20

'Set the stop frequency to 20 kHz
APx.SteppedFrequencySweep.Generator.StopFrequency.Value = 20000

'Set the number of sweep points to 11
APx.SteppedFrequencySweep.Generator.SweepParameters.NumberOfPoints = 11
```

```
'Set the sweep step spacing to logarithmic
APx.SteppedFrequencySweep.Generator.SweepParameters.StepType = SweepStepType.Log
```

C#

```
//Set the start frequency to 20 Hz
APx.SteppedFrequencySweep.Generator.StartFrequency.Value = 20;

//Set the stop frequency to 20 kHz
APx.SteppedFrequencySweep.Generator.StopFrequency.Value = 20000;

//Set the number of sweep points to 11
APx.SteppedFrequencySweep.Generator.SweepParameters.NumberOfPoints = 11;

//Set the sweep step spacing to logarithmic
APx.SteppedFrequencySweep.Generator.SweepParameters.StepType = SweepStepType.Log;
```

Controlling analysis settings (Sequence Mode)

Some measurements in the APx software have additional analysis parameters such as low-pass, high-pass, and weighting filter selections.

The following example shows how to configure the filter settings for the THD+N measurement.

VB.NET

```
'Enable the low-pass filter setting
APx.ThdN.LowpassFilterFrequencyEnabled = True

'Set the low-pass filter frequency to 18 kHz
APx.ThdN.LowpassFilterFrequency.Value = 18000

'Enable the high-pass filter setting
APx.ThdN.HighpassFilterFrequencyEnabled = True

'Set the high-pass filter frequency to 250 Hz
APx.ThdN.HighpassFilterFrequency.Value = 250

'Enable the noise weighting setting
APx.ThdN.NoiseWeightingEnabled = True

'Set the noise weighting filter to "A-weighting"
APx.ThdN.NoiseWeighting = WeightingFilterType.wt_A
```

C#

```
//Enable the low-pass filter setting
APx.ThdN.LowpassFilterFrequencyEnabled = true;

//Set the low-pass filter frequency to 18 kHz
APx.ThdN.LowpassFilterFrequency.Value = 18000;

//Enable the high-pass filter setting
APx.ThdN.HighpassFilterFrequencyEnabled = true;
```

```
//Set the high-pass filter frequency to 250 Hz
APx.ThdN.HighpassFilterFrequency.Value = 250;

//Enable the noise weighting setting
APx.ThdN.NoiseWeightingEnabled = true;

//Set the noise weighting filter to "A-weighting"
APx.ThdN.NoiseWeighting = WeightingFilterType.wt_A;
```

Running measurements and getting results (Sequence Mode)

Some measurements in APx take many readings per second. Examples include Level and Gain, THD+N, and Interchannel Phase. Other measurements are batch measurements. The measurement starts, readings are taken, and the measurement stops. Examples include sweeps like Frequency Response and many crosstalk measurements.

For measurements which take many readings per second, the way to get a settled reading is to run the measurement via the sequencer. All measurements can be run via the sequencer to ensure that readings are settled.

Getting settled meter readings

The following example shows getting settled meter readings for the Level and Gain measurement.

VB.NET

```
Try
	'Get the Level and Gain measurement from the sequence
	Dim LevelAndGain As ISequenceMeasurement = APx.Sequence.GetMeasurement("Signal
Path1", "Level and Gain")

	'Make sure the measurement is checked
	LevelAndGain.Checked = True

	'Run the measurement and record settled results
	LevelAndGain.Run()

	'Check to see that the measurement sequence ran correctly
	If LevelAndGain.HasSequenceResults Then
		'Get an array settled meter readings
		Dim readingValues As Double()
		readingValues =
LevelAndGain.SequenceResults(MeasurementResultType.RmsLevelMeter).GetMeterValues()
	End If
Catch ex As Exception
	'An error occurred such as Digital Unlock.  Report the error to the user
	MessageBox.Show(ex.Message)
End Try
```

C#

```
try
{
    //Get the Level and Gain measurement from the sequence
```

```
    ISequenceMeasurement LevelAndGain;
    LevelAndGain = APx.Sequence.GetMeasurement("Signal Path1", "Level and Gain");

    //Make sure the measurement is checked
    LevelAndGain.Checked = true;

    //Run the measurement and record settled results
    LevelAndGain.Run();

    //Check to see that the measurement sequence ran correctly
    if(LevelAndGain.HasSequenceResults)
    {
        //Get an array settled meter readings
        double[] readingValues;
        readingValues =
LevelAndGain.SequenceResults[MeasurementResultType.RmsLevelMeter].GetMeterValues();
    }
}
catch(Exception ex)
{
    //An error occurred such as Digital Unlock.  Report the error to the user
    MessageBox.Show(ex.Message);
}
```

Getting X,Y graph results

Measurements which produce X,Y graph data, such as Signal Analyzer can also be run via the sequencer. The X,Y data values can be accessed via the API.

VB.NET

```
Try
       'Get the Signal Analyzer measurement from the sequence
       Dim SignalAnalzyer As ISequenceMeasurement = APx.Sequence.GetMeasurement("Signal
Path1", "Signal Analyzer")

       'Make sure the measurement is checked
       SignalAnalzyer.Checked = True

       'Run the measurement and record settled results
       SignalAnalzyer.Run()

       'Check to see that the measurement sequence ran correctly
       If SignalAnalzyer.HasSequenceResults Then
              'Get the sequence result data for the FFT spectrum graph
              Dim SequenceResults As ISequenceResult
              SequenceResults =
SignalAnalzyer.SequenceResults(MeasurementResultType.FFTSpectrum)

              'Get the x-axis values
              Dim xValues As Double()
              xValues = SequenceResults.GetXValues(InputChannelIndex.Ch1)

              'Get the y-axis values
              Dim yValues As Double()
              yValues = SequenceResults.GetYValues(InputChannelIndex.Ch1)
```

```
        End If
Catch ex As Exception
        'An error occurred such as Digital Unlock.  Report the error to the user
        MessageBox.Show(ex.Message)
End Try
```

C#

```
try
{
    //Get the Signal Analyzer measurement from the sequence
    ISequenceMeasurement SignalAnalzyer = APx.Sequence.GetMeasurement("Signal Path1",
"Signal Analyzer");

    //Make sure the measurement is checked
    SignalAnalzyer.Checked = true;

    //Run the measurement and record settled results
    SignalAnalzyer.Run();

    //Check to see that the measurement sequence ran correctly
    if(SignalAnalzyer.HasSequenceResults)
    {
        //Get the sequence result data for the FFT spectrum graph
        ISequenceResult SequenceResults;
        SequenceResults =
SignalAnalzyer.SequenceResults[MeasurementResultType.FFTSpectrum];

        //Get the x-axis values
        double[] xValues;
        xValues = SequenceResults.GetXValues(InputChannelIndex.Ch1);

        //Get the y-axis values
        double[] yValues;
        yValues = SequenceResults.GetYValues(InputChannelIndex.Ch1);
    }
}
catch(Exception ex)
{
    //An error occurred such as Digital Unlock.  Report the error to the user
    MessageBox.Show(ex.Message);
}
```

Controlling signal path settings (Bench Mode)

It is often useful to configure the input or output connector and the settings associated with the selected connector. The following example demonstrates setting the input connector to “Analog Balanced”, and configuring the some of the input settings.

VB.NET

```
'Set the input connector to Analog Balanced
APx.BenchMode.Setup.InputConnector.Type = InputConnectorType.AnalogBalanced

'Set the number of input channels to 2
APx.BenchMode.Setup.AnalogInputChannelCount = 2
```

```
'Set the analog ADC bandwidth
APx.BenchMode.Setup.LowpassFilterAnalog = LowpassFilterModeAnalog.AdcPassband
APx.BenchMode.Setup.LowpassFilterAnalogBandwidth = AdcBandwidth.bw90k

'Set the input termination to 600 ohm (for 52x instruments)
APx.BenchMode.Setup.AnalogInput.SetTermination(InputChannelIndex.Ch1,
                                AnalogInputTermination.InputTermination_600)
```

C#

```
//Set the input connector to Analog Balanced
APx.BenchMode.Setup.InputConnector.Type = InputConnectorType.AnalogBalanced;

//'Set the number of input channels to 2
APx.BenchMode.Setup.AnalogInputChannelCount = 2;

//Set the analog ADC bandwidth
APx.BenchMode.Setup.LowpassFilterAnalog = LowpassFilterModeAnalog.AdcPassband;
APx.BenchMode.Setup.LowpassFilterAnalogBandwidth = AdcBandwidth.bw90k;

//Set the input termination to 600 ohm (for 52x instruments)
APx.BenchMode.Setup.AnalogInput.SetTermination(InputChannelIndex.Ch1,
                                AnalogInputTermination.InputTermination_600);
```

Controlling generator settings (Bench Mode)

Bench Mode has a single generator which has many configuration options. All of the settings in the user interface generator panels are available via the API. This means settings such as Frequency, Level, and Waveform can all be configured.

Setting generator level and frequency

The following example configures the generator for Bench Mode.

VB.NET

```
'Set the first channel of the analog generator to 100.0 mVrms
APx.BenchMode.Generator.Levels.SetValue(OutputChannelIndex.Ch1, 0.1)

'Set the generator frequency to 2.000 kHz
APx.BenchMode.Generator.Frequency.Value = 2000
```

C#

```
'Set the first channel of the analog generator to 100.0 mVrms
APx.BenchMode.Generator.Levels.SetValue(OutputChannelIndex.Ch1, 0.1);

'Set the generator frequency to 2.000 kHz
APx.BenchMode.Generator.Frequency.Value = 2000;
```

Selecting a different generator type

The Bench Mode generator supports different waveform types such as “Sine”, “Square”, “Sine, Dual”, and “Sine, Var Phase”. The generator type can be accessed via the `.Waveform` property. The available generator types can be enumerated using the `.WaveformNames` property.

The following example shows setting the Bench Mode generator to “Sine, Dual”. If the specified waveform is spelled incorrectly, or is not supported, APx will throw an exception indicating the waveform selection is not valid. The selection is not case-sensitive.

VB.NET

```
'Set the generator waveform to Sine, Dual
APx.BenchMode.Generator.Waveform = "Sine, Dual"

'Set the Frequency A value
APx.BenchMode.Generator.Frequency.Value = 1000.0

'Set the Frequency B value
APx.BenchMode.Generator.SplitFrequency.FrequencyB.Value = 4000.0
```

C#

```
//Set the generator waveform to Sine, Dual
APx.LevelAndGain.Generator.Waveform = "Sine, Dual";

//Set the Frequency A value
APx.LevelAndGain.Generator.Frequency.Value = 1000.0;

//Set the Frequency B value
APx.LevelAndGain.Generator.SplitFrequency.FrequencyB.Value = 4000.0;
```

Loading and selecting a waveform

Many generators in APx can play a waveform from a file. Supported linear waveform types include .WAV, .AGM, .AGS. APx can also play encoded audio waveforms when the output connector is digital. Supported waveforms include many Dolby® and DTS® file formats.

Once a waveform file has been loaded into a generator, it is available for use by all other measurements in the project. The list of available waveform files for any measurement is accessed via the `.WaveformNames` property of the generator, or via the `APx.AttachedProjectItems API`.

VB.NET

```
'Select and load a file called Test1.wav for playback
APx.BenchMode.Generator.LoadWaveformFile("c:\WAV\Test1.wav")

'Select and load a file called Test2.wav for playback
APx.BenchMode.Generator.LoadWaveformFile("c:\WAV\Test2.wav")

'Re-select Test1.wav
APx.BenchMode.Generator.Waveform = "Test1.wav"
```

C#

```
//Select and load a file called Test1.wav for playback
APx.BenchMode.Generator.LoadWaveformFile(@"c:\WAV\Test1.wav");

//Select and load a file called Test2.wav for playback
APx.BenchMode.Generator.LoadWaveformFile(@"c:\WAV\Test2.wav");

//Re-select Test1.wav
APx.BenchMode.Generator.Waveform = "Test1.wav";
```

Controlling sweep settings

The sweep in Bench Mode can be configured to control different parameters of the APx system. The `.Source` property controls the primary sweep setting. The sweep also offers nesting, meaning a secondary parameter can be modified for each cycle of the primary sweep. The `.NestedSweep.Source` parameter controls the nested sweep setting. The primary and nested sweep parameters are all available via the API.

VB.NET

```
With APx.BenchMode.Measurements.SteppedSweep
       'Set the primary sweep parameter to Generator Frequency
       .Source = SweepSourceParameterType.GeneratorFrequency
       'Set the start frequency to 20 kHz
       .SourceParameters.Start.Value = 20000.0
       'Set the stop frequency to 20 Hz
       .SourceParameters.Stop.Value = 20.0
       'Set the number of sweep points to 11
       .SourceParameters.NumberOfPoints = 11
       'Set the sweep step spacing to logarithmic
       .SourceParameters.StepType = SweepStepType.Log

       'Set the nested sweep parameter to Generator Level
       .NestedSweep.Source = Sweep2ParameterType.GeneratorLevel
       'Set the start level to 100 mVrms
       .NestedSweep.Source2Parameters.Start.Value = 0.1
       'Set the stop level to 500 mVrms
       .NestedSweep.Source2Parameters.Stop.Value = 0.5
       'Set the number of sweep points to 3
       .NestedSweep.Source2Parameters.NumberOfPoints = 3
       'Set the sweep step spacing to logarithmic
       .NestedSweep.Source2Parameters.StepType = SweepStepType.Log
End With
```

C#

```
ISteppedSweep sweep = APx.BenchMode.Measurements.SteppedSweep;
//Set the primary sweep parameter to Generator Frequency
sweep.Source = SweepSourceParameterType.GeneratorFrequency;
//Set the start frequency to 20 kHz
sweep.SourceParameters.Start.Value = 20000.0;
//Set the stop frequency to 20 Hz
sweep.SourceParameters.Stop.Value = 20.0;
//Set the number of sweep points to 11
sweep.SourceParameters.NumberOfPoints = 11;
```

```
//Set the sweep step spacing to logarithmic
sweep.SourceParameters.StepType = SweepStepType.Log;

//Set the nested sweep parameter to Generator Level
sweep.NestedSweep.Source = Sweep2ParameterType.GeneratorLevel;
//Set the start level to 100 mVrms
sweep.NestedSweep.Source2Parameters.Start.Value = 0.1;
//Set the stop level to 500 mVrms
sweep.NestedSweep.Source2Parameters.Stop.Value = 0.5;
//Set the number of sweep points to 3
sweep.NestedSweep.Source2Parameters.NumberOfPoints = 3;
//Set the sweep step spacing to logarithmic
sweep.NestedSweep.Source2Parameters.StepType = SweepStepType.Log;
```

Controlling analysis settings (Bench Mode)

Bench mode analyzer parameters such as notch filter tuning are settable via the API.

VB.NET

```
'Set notch filter tuning to a fixed frequency value
APx.BenchMode.Analyzer.NotchFilterTuning = FilterTuningType.FixedFrequency
'Set notch filter tuning frequency to 2.5 kHz
APx.BenchMode.Analyzer.NotchFixedFilterFrequency.Value = 2500.0
```

C#

```
//Set notch filter tuning to a fixed frequency value
APx.BenchMode.Analyzer.NotchFilterTuning = FilterTuningType.FixedFrequency;
//Set notch filter tuning frequency to 2.5 kHz
APx.BenchMode.Analyzer.NotchFixedFilterFrequency.Value = 2500.0;
```

Running measurements and getting results (Bench Mode)

In Bench Mode, meter readings are instantaneous and are updated at the system reading rate. However, settled values can be acquired via the API. Batch measurements like the Sweep and FFT measurements require the system to make one or more data acquisitions and process the data before producing results. In Bench Mode, batch measurements do not block, so polling of the measurement completion status is required.

Getting settled meter readings

The following example shows getting settled meter readings for RMS Level and THD+N Ratio values.

VB.NET

```
Try
      'Ask APx to make settled readings for RMS Level and THD+N Ratio
      Dim results As ISettledResultCollection =
            APx.BenchMode.GetSettledMeterReadings(SettlingMeterType.RmsLevel,
                                                  SettlingMeterType.ThdNRatio)

      'Get the settled readings for the RMS Level meter
      Dim level As Double() = results(SettlingMeterType.RmsLevel).GetValues("Vrms")
      'Get the settled readings for the THD+N ratio meter
      Dim thdN As Double() = results(SettlingMeterType.ThdNRatio).GetValues("dB")
```

```
Catch ex As Exception
       'An error occurred such as Digital Unlock.  Report the error to the user
       MessageBox.Show(ex.Message)
End Try
```

C#

```
try
{
       //Ask APx to make settled readings for RMS Level and THD+N Ratio
       ISettledResultCollection results =
            APx.BenchMode.GetSettledMeterReadings(SettlingMeterType.RmsLevel,
                                                  SettlingMeterType.ThdNRatio);

       //Get the settled readings for the RMS Level meter
       double[] level = results[SettlingMeterType.RmsLevel].GetValues("Vrms");
       //Get the settled readings for the THD+N ratio meter
       double[] thdN = results[SettlingMeterType.ThdNRatio].GetValues("dB");
}
catch (Exception ex)
{
       //An error occurred such as Digital Unlock.  Report the error to the user
       MessageBox.Show(ex.Message);
}
```

Getting X,Y graph results

Measurements such as the FFT measurement in Bench Mode do not block the caller while they run. This example demonstrates running the FFT measurement and waiting for it to finish, and then extracting the X,Y values for a specific channel.

VB.NET

```
'Make the FFT measurement visible
APx.BenchMode.Measurements.Fft.Show()
With APx.BenchMode.Measurements
       Try
           'Start the FFT measurement
           .Fft.Start()
           While .Fft.IsStarted
              'Wait for it to finish
              System.Threading.Thread.Sleep(100)
           End While
           'get the X and Y values for Ch1
           Dim xValues As Double() = .Fft.FFTSpectrum.GetXValues(InputChannelIndex.Ch1)
           Dim yValues As Double() = .Fft.FFTSpectrum.GetYValues(InputChannelIndex.Ch1)
       Catch ex As Exception
           'An error occurred.  Report it to the user
           MessageBox.Show(ex.Message)
       End Try
End With
```

C#

```
//Make the FFT measurement visible
```

```
APx.BenchMode.Measurements.Fft.Show();
IFftAnalyzer fft = APx.BenchMode.Measurements.Fft;
try
{
	//Start the FFT measurement
	fft.Start();
	while (fft.IsStarted)
	{
		//Wait for it to finish
		System.Threading.Thread.Sleep(100);
	}
	//get the X and Y values for Ch1
	double[] xValues = fft.FFTSpectrum.GetXValues(InputChannelIndex.Ch1);
	double[] yValues = fft.FFTSpectrum.GetYValues(InputChannelIndex.Ch1);
}
catch (Exception ex)
{
	//An error occurred.  Report it to the user
	MessageBox.Show(ex.Message);
}
```

Working with primary and derived results

When measurements are first created, they have a standard set of meter, XY graph, or other results associated with the measurement. After the measurement has been created, additional views of the data can be added. For example, a user may want to view THD+N Ratio in dB units or in % units. To accomplish this, the user could add two THD+N Ratio results to the THD+N measurement. Both results are considered Primary results.

APx also allows computations to be performed on result data. Computations are added using Derived results. For example, a user may want to smooth frequency response data or normalize the data at a specified reference frequency.

The .Graphs property

Each measurement in the API has a `.Graphs` property which allows the programmer to enumerate all of the results within a particular measurement. This property returns an `IGraphCollection` object which has the ability to index into the collection by integer index or by result name. If two results have the same name, i.e. “RMS Level”, the first instance is returned. As with measurement and signal path names, it is recommended that each result have a unique name so it can be addressed via the API.

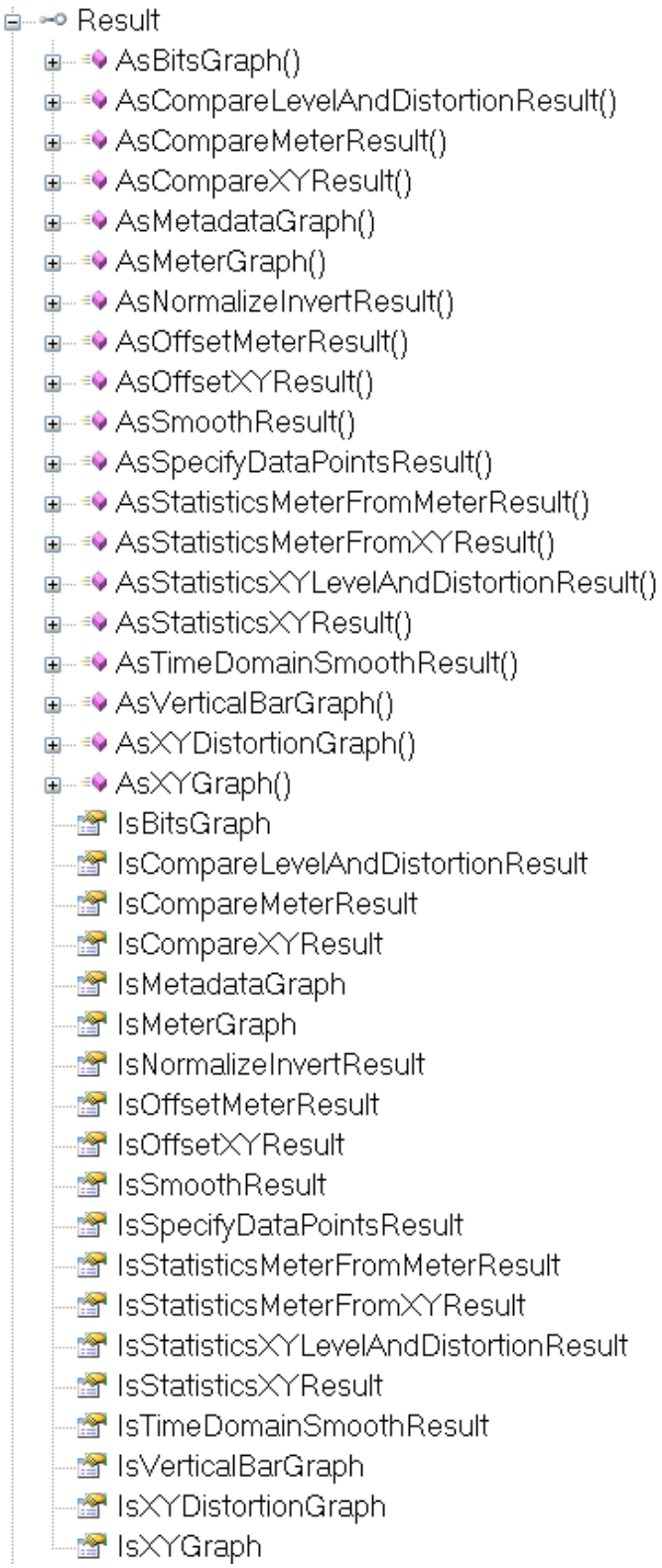
Deleted results

Users can also delete results from the measurement. As such, there is no guarantee from the API that a particular result is available from a measurement. For example, if the user has deleted the “RMS Level” result from the “Level and Gain” measurement, then the `APx.LevelAndGain.Level` property will attempt to access a result which is not present. In this case, the API will throw an `APException` with the `ErrorCode` property indicating `APError.InvalidMeasurementResultType`.

IDynamicResultGraph

Each graph in the `IGraphCollection` object is of type `IGraph`. There is a property on the `IGraph` object called `Result` which returns an object of type `IDynamicResultGraph`. This object allows the programmer to determine the type of the result. This object also allows the programmer to access the specific type parameters of the result.

The `IDynamicResultGraph` object has properties to determine the type of the result. The properties are in the form of a Boolean property with the naming convention "IsXXX" where "XXX" is a specific result type. The object has methods with the naming convention "AsXXX" which return an object of the specified type. See table below.



The result has properties to determine the type of the result, as well as methods to return an object of the requested type.

If the `IsXXX` property returns false, then the `AsXXX` propety will throw an `APException`.

Is/As example

The following example code shows the difference between accessing the “Level” graph in a “Frequency Response” measurement via the standard API, and via the `.Graphs` API.

VB.NET

```
Dim XValues As Double()
'If you know that the Frequency Response measurement contains a Level result
'the following API call will allow you to 'extract the Level X axis values
XValues = APx.FrequencyResponse.Level.GetXValues(InputChannelIndex.Ch1)

'If you do not know exactly which results are present, use the .Graphs
'property to figure out what is present
For Each Graph As IGraph In APx.FrequencyResponse.Graphs
      'Found a graph which is an XY graph
      If Graph.Result.IsXYGraph Then
            'Found a Level vs Frequency graph
            If Graph.ViewType = MeasurementResultType.LevelVsFrequency Then
                  'Get the X axis values
                  Dim XYGraph As IXYGraph
                  XYGraph = Graph.Result.AsXYGraph()
                  XValues = XYGraph.GetXValues(InputChannelIndex.Ch1)
            End If
      End If
Next
```

C#

```
double[] XValues;
//If you know that the Frequency Response measurement contains a Level result
//the following API call will allow you to
//extract the Level X axis values
XValues = APx.FrequencyResponse.Level.GetXValues(InputChannelIndex.Ch1);

//If you do not know exactly which results are present, use the .Graphs
//property to figure out what is present
foreach(IGraph Graph in APx.FrequencyResponse.Graphs)
{
    //Found a graph which is an XY graph
    if(Graph.Result.IsXYGraph)
    {
        //Found a Level vs Frequency graph
        if(Graph.ViewType == MeasurementResultType.LevelVsFrequency)
        {
            //Get the X axis values
            IXYGraph XYGraph;
            XYGraph = Graph.Result.AsXYGraph();
            XValues = XYGraph.GetXValues(InputChannelIndex.Ch1);
        }
    }
}
```

Derived results

Results which derive a computed value from another result are accessible via the `.Graphs` property. Use the `IDynamicResultGraph` interface to access the derived result properties.

Derived results example

The following example shows how to access the `OctaveSmoothing` property of a Smooth derived result.

VB.NET

```
'When accessing derived results, the API does not offer a property for
'finding the result.  Use the .Graphs property to find the result
For Each Graph As IGraph In APx.FrequencyResponse.Graphs
      'Ask if this result is a Smoothed result
      If Graph.Result.IsSmoothResult Then
            'Found a Smoothed result
            Dim Smooth As ISmoothResult
            Smooth = Graph.Result.AsSmoothResult()
            'Set the smoothing type
            Smooth.OctaveSmoothing = OctaveSmoothingType.Octave3
      End If
Next
```

C#

```
//When accessing derived results, the API does not offer a property for
//finding the result.  Use the .Graphs property to find the result
foreach(IGraph Graph in APx.FrequencyResponse.Graphs
{
    //Ask if this result is a Smoothed result
    if(Graph.Result.IsSmoothResult)
    {
        //Found a Smoothed result
        ISmoothResult Smooth;
        Smooth = Graph.Result.AsSmoothResult();
            //Set the smoothing type
        Smooth.OctaveSmoothing = OctaveSmoothingType.Octave3;
    }
}
```

Advanced topics

Optimizing performance

There are a few tricks to making APx500 API programs run faster. The first is to turn off the Signal Monitors. The other is to “park” APx on a measurement which is not updating measurement values continuously.

VB.NET

```
'Turn off the signal monitors
APx.SignalMonitorsEnabled = False

'Show a measurement which does not update automatically
APx.ShowMeasurement("Signal Path1", "Frequency Response")
```

C#

```
//Turn off the signal monitors
```

```
APx.SignalMonitorsEnabled = false;

//Show a measurement which does not update automatically
APx.ShowMeasurement("Signal Path1", "Frequency Response");
```

Additional Resources and Sample Programs

Audio Precision has provided APx programming examples and help files on the Resource Disc, available on the Audio Precision website: http://www.ap.com/download/api

You can also visit the APx programming web page: http://ap.com/products/apx/automation

You can also contact Audio Precision Technical Support for further assistance: techsupport@audioprecision.com