Controlling APx500 Plugin Measurements Via the API

Audio Precision — Tech Guide // Benjamin Kilada

TECH GUIDE

Summary

The APx500 software comes with many built-in measurements. Additional measurements, known as plugin measurements, can be added to the software by running installers made available on the Audio Precision (AP) website. Plugin measurements integrate directly with the APx500 software, allowing users to take advantage of built-in APx500 features such as pass/fail limits, sequencer control, derived results, and reporting.

Like built-in measurements, plugin measurements can be controlled programmatically via the software's Application Programming Interface (API). However, unlike built-in measurements, all plugin measurements derive from the Signal Acquisition measurement and are controlled via the Signal Acquisition APIs. This document provides an overview of how to do this.

Note: All API examples are provided in the Visual C# programming language. For the example code to function, the related plugin must first be installed on the PC on which the code is executing.

The APx500 Plugin Framework

In the 4.2 release of the APx500 software, AP added a preliminary plugin framework. This framework consists of infrastructure within the APx software that allows custom measurements and analysis features to be added independent of an official APx software release. The first AP product to take advantage of this framework was the Speech Transmission Index (STI) plugin, introduced in October 2015. Further enhancements to the plugin framework were added in the APx 4.3 software release, and then again in APx500 v4.5, when the application transitioned from 32-bit to 64-bit.

Note: Plugins of version "2.x" are 64-bit and are incompatible with APx500 software version 4.4 and earlier.

The plugin framework allows for both custom measurements and custom derived results to be added to the APx500 software. (In this context, custom refers to measurements and derived results that are not natively part of the APx500 software).

Custom measurements derive from the Signal Acquisition measurement and typically have some of the same generator and analyzer controls (for example, generator level or acquisition length) and primary results (Acquired Waveform). In addition, plugin measurements may have their own unique generator and analyzer controls (custom controls) and primary results (custom results). The controls and results that are shared with Signal Acquisition can be accessed using standard Signal Acquisition APIs, whereas custom controls and results can be modified by indexing into a collection of properties using the control or result name. This is described with examples in the following sections.

Custom derived results can be of type meter or XY and often have custom controls (controls that appear below a derived result graph) that can be accessed using "get" and "set" APIs. This is described in sections eight and nine.

1. Installing a Plugin Measurement

Download and install the desired plugin(s) using the installer(s) provided on the AP website, ensuring that the APx500 application is first closed. Plugin compatibility with APx500 software versions is discussed on the download page of each plugin.

2. Initiating the API Reference

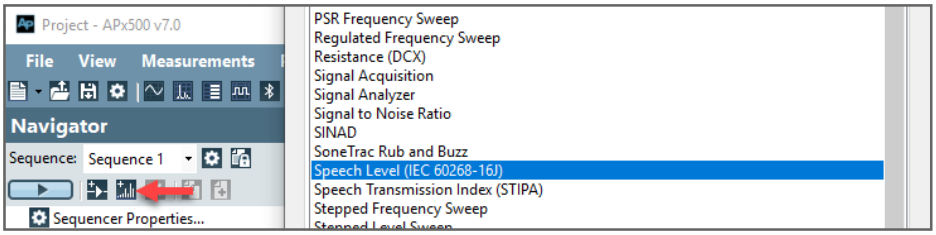
When controlling a plugin measurement from the API, it is necessary to first instantiate the APx500 class that represents the APx500 software application. Doing so will launch the APx500 software if closed.
To access this class, you must first reference the Audio Precision API Dynamic Link Library (DLL) that is installed with the APx500 application. For more information on referencing the API DLL, please refer to the materials provided in the API developer tools, available on the AP website.

Note that in most cases, it is necessary to initialize the application into Sequence Mode, as most plugins are only available in this operating mode. An example of how to do this is shown below.

```
APx500 APx = new APx500(APxOperatingMode.SequenceMode);
```

3. Adding a Measurement

To add a plugin measurement to the current project, the add measurement method must be called using the string name of the measurement as the second argument. Plugin measurement names can be found in the add-measurement dialog in the APx500 graphical user interface (GUI). This dialog can be opened using the button highlighted in the image shown below.



An example of how to add the STI Speech Level measurement (highlighted in the image above) is shown below.

```
APx.AddMeasurement(0, "Speech Level (IEC 60268-16J)");
```

Note: Adding a measurement also makes it visible.

4. Showing a Measurement

To show a measurement that already exists in the project, the “show measurement” method must be called using the string name of the plugin measurement as demonstrated below.

```
APx.ShowMeasurement(0, “Speech Level (IEC 60268-16J)”);
```

With the plugin measurement added and visible, we can now modify the measurement settings.

5. Generator Settings

The Signal Acquisition measurement has several built-in generator settings that can be configured via the API. These settings include:

- Waveform
- Level
- Offset
- Loop Waveform
- Channel enable/disable

When these settings are shown in a plugin measurement, they can be get or set via the built-in APx.SignalAcquisition.Generator APIs, which are discoverable via the API browser application. An example of how to use these APIs is shown below.

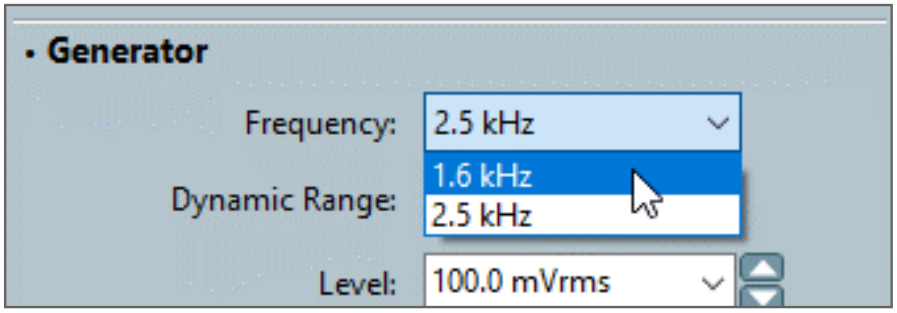
```
APx.SignalAcquisition.Generator.Waveform = “Default 48 kHz.wav”;
APx.SignalAcquisition.Generator.Levels.SetValue(OutputChannelIndex.Ch1, “1 Vrms”);
```

Generator settings that are unique to plugin measurements (and not available in Signal Acquisition) can be get or set via the Signal Acquisition “GeneratorProperties” collection. To do so, the appropriate get or set method must be called based on the type of control that is being modified. A breakdown of when to use each type of method is included below.

Note: All set methods take the string name of the control as the first argument, and the value to be set as the second argument. Get methods take only the string name of the control as the argument.

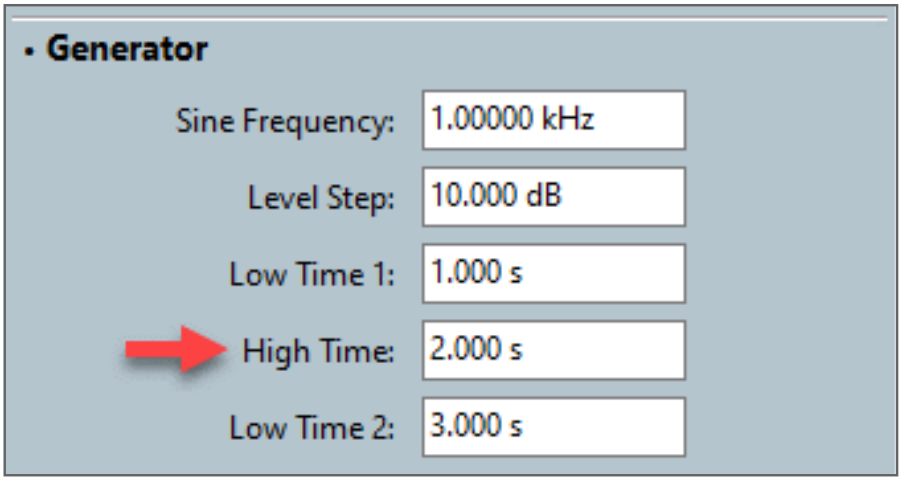
A **drop-down pick list** is of type string and can be set to any value in the pick list via the SetString method. Similarly, the current value can be obtained using the GetString method. For example, the Frequency control in the Attack and Recovery plugin measurement can be get and set in the following manner:

```
string freq = APx.SignalAcquisition.GeneratorProperties.GetString("Frequency");
APx.SignalAcquisition.GeneratorProperties.SetString("Frequency", "1.6 kHz");
```



A **numerical field that accepts fractional numbers** is of type double and can be set to any whole or fractional number that is within the range of accepted values via the SetDouble method. Similarly, the current value can be obtained using the GetDouble method. For example, the High Time control in the Attack and Release measurement can be get and set in the following manner:

```
double highTime = APx.SignalAcquisition.GeneratorProperties.GetDouble("High Time");
APx.SignalAcquisition.GeneratorProperties.SetDouble("High Time", 1.5);
```



Note: There are currently no plugin measurements with generator properties of type Integer, Bool, Bool Array or Enum.

6. Analyzer Settings

The Signal Acquisition measurement has several built-in analyzer settings that can be configured via the API. These settings include:

- Acquisition Type
- Acquisition Length
- Trigger Type
- Delay Time
- Save To File

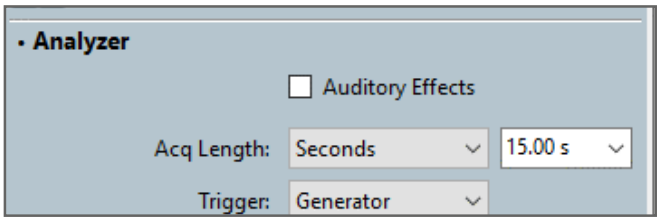
When these settings are shown in a plugin measurement, they can be modified via the built-in APx.SignalAcquisition APIs, discoverable via the API browser application. An example of how to use these APIs is shown below.

```
APx.SignalAcquisition.AcquisitionType = AcqLengthType.Seconds;
APx.SignalAcquisition.AcquisitionSeconds = 10;
APx.SignalAcquisition.TriggerType = TriggerType.Generator;
APx.SignalAcquisition.DelayTimeInSeconds = 0.1;
APx.SignalAcquisition.SaveToFileSettings.SaveAcquisitionToFile = true;
```

Analyzer settings that are unique to a plugin measurement (and not available in Signal Acquisition) can be get or set via the Signal Acquisition properties collection. As with generator properties, the appropriate get or set method must be called based on the type of control that is being modified. A breakdown of when to use each type of method is included below.

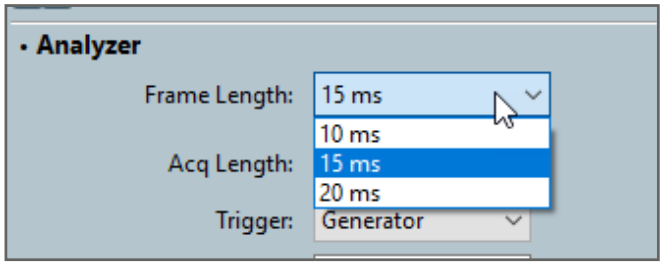
A **checkbox** is of type boolean and can be set to either true (checked) or false (unchecked) via the SetBool method. Similarly, a checkbox value can be obtained using the GetBool method. For example, the Auditory Effects checkbox in the Speech Transmission Index (STI) measurement can be get and set in the following manner:

```
bool useEffects = APx.SignalAcquisition.Properties.GetBool("Auditory Effects");
APx.SignalAcquisition.Properties.SetBool("Auditory Effects", true);
```



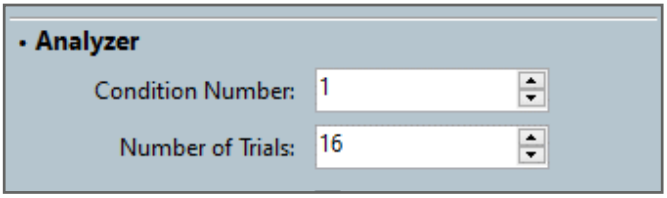
A **drop-down pick list** is of type string and can be set to any value in the pick list via the SetString method. Similarly, the value can be obtained using the GetString method. For example, the Frame Length control in the STI Speech Level plugin measurement can be get and set in the following manner:

```
string frameLength = APx.SignalAcquisition.Properties.GetString("Frame Length", "20 ms");
APx.SignalAcquisition.Properties.SetString("Frame Length", "20 ms");
```



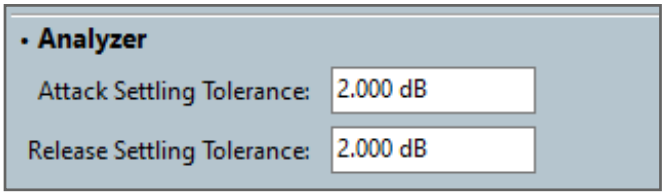
A **numerical field with no decimal places** is of type integer and can be set to any whole number that is within the range of accepted values via the SetInt method. Similarly, the current integer value can be obtained using the GetInt method. For example, the Number of Trials control in the ABC-MRT plugin measurement can be get and set in the following manner:

```
int numTrials = APx.SignalAcquisition.Properties.GetInt("Number of Trials");
APx.SignalAcquisition.Properties.SetInt("Number of Trials", 16);
```



A **numerical field that accepts fractional numbers** is of type double and can be set to any whole or fractional number that is within the range of accepted values via the SetDouble method. Similarly, the value can be obtained using the GetDouble method. For example, the Attack Settling Tolerance control in the Attack and Release measurement can be get and set in the following manner:

```
double tol = APx.SignalAcquisition.Properties.GetDouble("Attack Settling Tolerance");
APx.SignalAcquisition.Properties.SetDouble("Attack Settling Tolerance", 1.5);
```



Note: There are currently no plugin measurements with analyzer properties of type Bool Array or Enum.

7. Results

Results in a plugin measurement can be modified by indexing into the Signal Acquisition graphs collection via the string name of the graph (or the integer index). For example, the Speech Level measurement's Frame Levels result can be included in the active sequence using the following API.

```
APx.SignalAcquisition.Graphs["Frame Levels (A-weighted)"].Checked = true;
```

To access the settings that pertain to a particular result type (for example, y-axis settings on an XY graph), the result must first be cast to the appropriate type using the "Result.As[GraphType]" APIs. An example of how to cast a result to type XY graph and modify the y-axis unit is shown below.

```
IXYGraph frameLevelsXY = APx.SignalAcquisition.Graphs["Frame Levels (A-weighted)"].
Result.AsXYGraph();
frameLevelsXY.YAxis.Unit = "dBv";
```

Similarly, a meter graph can be cast using the AsMeterGraph() method shown below:

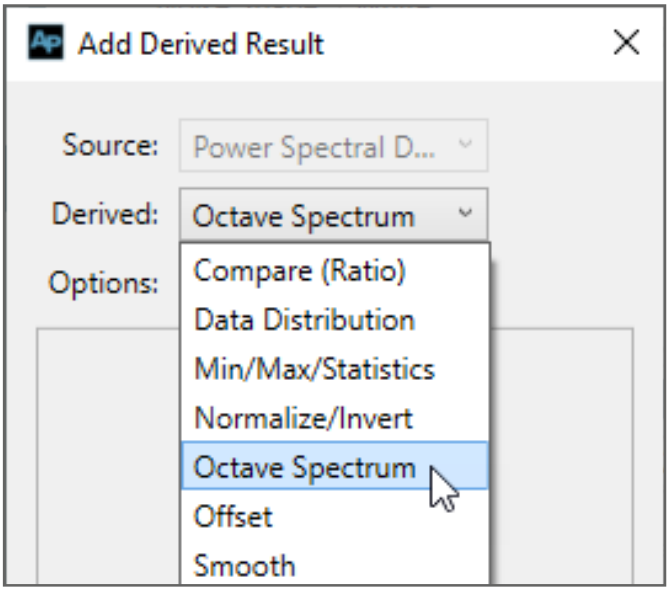
```
IMeterGraph rmsLevelMeter = APx.SignalAcquisition.Graphs["RMS Level (A-weighted)"].
Result.AsMeterGraph();
rmsLevelMeter.Axis.Unit = "dBV";
```

If the plugin has an Acquired Waveform result, the result can also be accessed using the standard Signal Acquisition APIs as shown below. No casting is necessary.

```
APx.SignalAcquisition.AcquiredWaveform.YAxis.Unit = "V";
```

8. Deriving a Custom Result

There are a few plugins that add derived result types to the picklists of measurement results. For example, installing the Octave Analysis plugin enables you to derive an “Octave Spectrum” result from a Power Spectral Density result in an FFT or Transfer Function measurement.



Plugin derived results can be added using the AddDerivedResult method by providing the string name of the derived result as the argument. An example of how to do so is shown below.

```
APx.BenchMode.Measurements.TransferFunction.PowerSpectralDensity.
AddDerivedResult(“Octave Spectrum”);
```

9. Custom Result Settings

Although the API shown above does add a derived result, a custom result's settings can only be accessed after casting the result to the appropriate custom processing result type. For example, the Octave Spectrum derived result (which is an XY graph) should be cast to type CustomProcessingXyResult. An example of how to do this is indicated below.

```
ICustomProcessingXyResult octaveSpectrum = APx.BenchMode.Measurements.
TransferFunction.PowerSpectralDensity.AddDerivedResult("Octave Spectrum").Result.
AsCustomProcessingXyResult();
```

Once cast to the appropriate type, a result's settings can be modified using its properties collection and the string name of the property. An example of how to get and set the Octave Bandwidth property of the Octave Spectrum result is shown below.

```
string bandwidth = octaveSpectrum.Properties.GetString("Octave Bandwidth");
octaveSpectrum.Properties.SetString("Octave Bandwidth", "1/12 Octave");
```

For more information on what get and set methods to use for each result setting, please refer to sections five and six, above.

© 2022 Audio Precision, Inc / All Rights Reserved.