

Vision™ MX280001A Spectrum Monitoring Application Programming Interface

MS2710xA, 9 kHz to 6 GHz

MS2090A, 9 kHz to 54 GHz

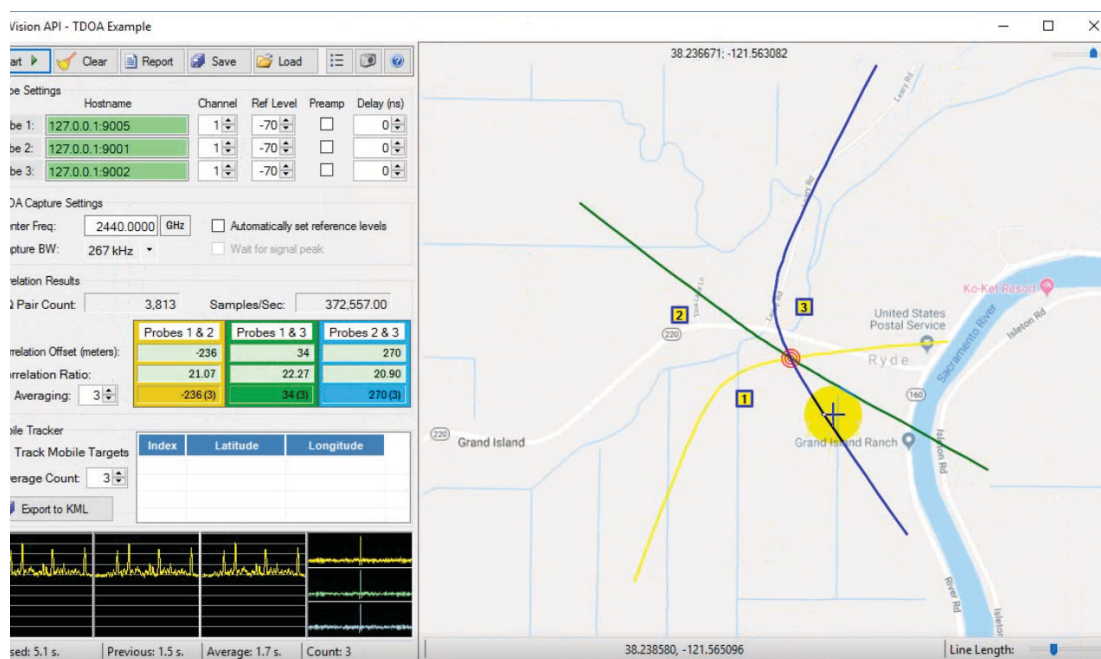


Table of Contents

Chapter 1—Overview

1-1	Introduction	1-1
	Additional Documentation	1-1
	Document Conventions	1-2
1-2	System Requirements	1-2
1-3	Using Vision API	1-2
	Frequency Settings	1-2
1-4	An Example Using the Vision API	1-3
	Example - Vision API Console Demo	1-4
1-5	Vision API User Controls Examples	1-5

Chapter 2—Vision User Controls

2-1	Introduction	2-1
2-2	File Viewer	2-1
	Trace Array	2-1
	Local File System	2-2
	Remote File System	2-2
	Channel Power Measurement (*.cpm) files.	2-3
2-3	TDOA Panel	2-4
	Basic User Interface (UI)	2-4
	Correlation Results and Map	2-5
	Correlation Results and IQ Graphs	2-5
	Moving Target Tracking	2-6
	Full User Interface	2-6
2-4	Radio Monitor	2-7
	Radio Monitor Icons	2-7
	Radio Monitor Buttons	2-8

Chapter 3—Vision API Reference

3-1	Introduction	3-1
3-2	Acquire Class	3-1
	Properties	3-1
	Acquire.ArchivePeriod Property	3-1
	Acquire.AutoArchive Property	3-1
	Acquire.AutoClearOld Property	3-2
	Acquire.Connected Property	3-2
	Acquire.Database Property	3-2
	Acquire.GetNotification Property	3-2
	Acquire.GetNotificationList Property	3-2
	Acquire.IsRunning Property	3-2
	Acquire.NotificationCount Property	3-3
	Acquire.SendNotifications Property	3-3
	Acquire.SweepInterval Property	3-3
	Acquire.SweepState Property	3-3

Table of Contents (Continued)

Acquire.ThreadCount Property	3-3
Methods	3-4
Acquire.AddChannel2Probe Method	3-4
Acquire.AddGroup Method	3-4
Acquire.AddNotification Method	3-4
Acquire.AddProbe2Database Method	3-5
Acquire.Archive Method	3-5
Acquire.ClearNotifications Method	3-5
Acquire.Compress Method	3-5
Acquire.ConfigureEmail Method	3-6
Acquire.CreateDatabase Method	3-6
Acquire.DeleteNotification Method	3-6
Acquire.LoadDatabase Method	3-7
Acquire.ModifyChannel Method	3-7
Acquire.ModifyProbeDefinition Method	3-7
Acquire.RemoveGroup Method	3-7
Acquire.RemoveProbeFromDatabase Method	3-8
Acquire.SetNotification Method	3-8
Acquire.StartSweeping Method	3-8
Acquire.StopSweeping Method	3-8
Acquire.SweepOnce Method	3-8
Acquire.TestNotification Method	3-8
Acquire.Trim Method	3-9
Acquire.ViewLog Method	3-9
Structures	3-9
ChannelInfo Structure	3-9
Fields	3-9
Active Field	3-9
Antenna Field	3-10
ChannelDelay Field	3-10
Direction Field	3-10
FFT Field	3-10
Frequency Field	3-10
MaskName Field	3-11
Preamplifier Field	3-11
RBW Field	3-11
RefLevel Field	3-11
RefLevelOffset Field	3-11
StartFreq Field	3-12
StopFreq Field	3-12
SweepTimeOut Field	3-12
TraceMode Field	3-12
TracePoints Field	3-12
VBW Field	3-12
Enumerations	3-13
ArchivePeriods Enumeration	3-13
NotifyType Enumeration	3-13
3-3 AngleOfArrival Class	3-14
Properties	3-14
AngleOfArrival.AntennaAngle Property	3-14
AngleOfArrival.DegreeMinuteSecond Property	3-14
AngleOfArrival.GPSCoordinates Property	3-14
AngleOfArrival.Hostname Property	3-14
AngleOfArrival.Location Property	3-15

Table of Contents (Continued)

AngleOfArrival.MeassuredPower Property	3-15
AngleOfArrival.ReferenceLevel Property	3-15
AngleOfArrival.SourceFrequency Property	3-15
AngleOfArrival.SourceMode Property	3-15
AngleOfArrival.StartFrequency Property	3-15
AngleOfArrival.StopFrequency Property	3-16
AngleOfArrival.Trace Property	3-16
Methods	3-16
AngleOfArrival.VerifySource Method	3-16
Structures	3-16
AOA_Result Structure	3-16
AntennaInfo Structure	3-16
Angle Field	3-16
Latitude Field	3-17
Longitude Field	3-17
Width Field	3-17
Direction Field	3-17
Height Field	3-17
Enumerations	3-18
PowerSource Enumeration	3-18
3-4 Database Class	3-19
Methods	3-19
Database.AddChannel Method	3-19
Database.AddProbe Method	3-19
Database.CreateEmptyDatabase Method	3-19
Database.DeleteChannel Method	3-19
Database.DeleteProbe Method	3-19
Database.GetChannel Method	3-20
Database.GetDatabasePath Method	3-20
Database.GetProbe Method	3-20
Database.GetTraces Method	3-21
Database.ModifyChannel Method	3-21
Database.ModifyProbe Method	3-21
Database.ProbeList Method	3-21
Database.SetDatabasePath Method	3-21
Structures	3-22
ChannelData Structure	3-22
ProbeDescription Structure	3-22
Fields	3-22
CarrierFreq Field	3-22
Channel Field	3-22
Failed Field	3-22
Passed Field	3-22
ProbeName Field	3-23
RefLevel Field	3-23
Sector Field	3-23
StartFreq Field	3-23
Status Field	3-23
StopFreq Field	3-23
Times Field	3-23
TraceCount Field	3-24
TraceTable Field	3-24
Traces Field	3-24

Table of Contents (Continued)

URL Field	3-24
Structures	3-24
ChannelDescription Structure	3-24
Fields	3-24
Active Field	3-24
Averaging Field	3-25
Channel Field	3-25
ChannelDelay Field	3-25
Direction Field	3-25
FFT Field	3-25
FailRate Field	3-25
Frequency Field	3-25
LastFail Field	3-26
LastSweep Field	3-26
MaskName Field	3-26
Preamp Field	3-26
ProbeName Field	3-26
RBW Field	3-26
RefLevel Field	3-26
RefLevelOffset Field	3-27
StartFreq Field	3-27
Status Field	3-27
StopFreq Field	3-27
TimeOut Field	3-27
TraceCount Field	3-27
TraceMode Field	3-27
TracePoints Field	3-28
VBW Field	3-28
ChannelCount Field	3-28
Contact Field	3-28
Description Field	3-28
ErrorCount Field	3-28
GroupName Field	3-28
Height Field	3-29
Hostname Field	3-29
LastFail Field	3-29
Latitude Field	3-29
Longitude Field	3-29
Name Field	3-29
Port Field	3-29
Status Field	3-30
TraceDatabase Field	3-30
TraceTable Field	3-30
3-5 FileViewer Class	3-31
Properties	3-31
FileViewer.CPMChannelDescriptionList Property	3-31
FileViewer.ChannelDescription Property	3-31
FileViewer.FileList Property	3-31
FileViewer.Filename Property	3-32
FileViewer.Folder Property	3-32
FileViewer.GetRemoteFolderList Property	3-32
FileViewer.Hostname Property	3-33
FileViewer.LocationType Property	3-33
FileViewer.RemoteFileList Property	3-33

Table of Contents (Continued)

	FileViewer.RemoteFolder Property	3-34
	FileViewer.SelectAllRemote Property	3-34
	FileViewer.ShowSidePanel Property	3-34
	FileViewer.ShowSpectrogram Property	3-35
	FileViewer.SnapShot Property	3-35
	Methods	3-35
	FileViewer.Close Method	3-35
	FileViewer.DownloadRemoteFiles Method	3-36
	FileViewer.SetTraceData Method	3-36
	Enumerations	3-37
	FileModes Enumeration	3-37
3-6	MaskEditor Class	3-38
	Properties	3-38
	MaskEditor.MaskName Property	3-38
	Methods	3-38
	MaskEditor.GetDatabasePath Method	3-38
	MaskEditor.GetMaskList Method	3-38
	MaskEditor.RemoveAMaskFromDB Method	3-38
	MaskEditor.SetDatabasePath Method	3-39
	MaskEditor.addSegmentedMaskToDB Method	3-39
	MaskEditor.addTraceMaskToDB Method	3-40
3-7	Monitor Class	3-41
	Monitor.Database()	3-41
	Monitor.TraceData()	3-41
	Monitor.TraceStartFrequency()	3-41
	Monitor.TraceStopFrquency()	3-41
	Monitor.TraceRBW()	3-41
	Monitor.TraceMarker()	3-41
3-8	Occupancy Class	3-42
	Properties	3-42
	Occupancy.AntennaPort Property	3-42
	Occupancy.ArchiveFolder Property	3-42
	Occupancy.ChannelWidth Property	3-42
	Occupancy.GroupByResult Property	3-42
	Occupancy.IncludeDetails Property	3-42
	Occupancy.Report Property	3-43
	Occupancy.StartFrequency Property	3-43
	Occupancy.StopFrequency Property	3-43
	Occupancy.Threshold Property	3-43
	Occupancy.ThresholdType Property	3-43
	Methods	3-44
	Occupancy.LoadOwnerList Method	3-44
3-9	PowerOnArrival Class	3-45
	Properties	3-45
	PowerOnArrival.DegreeMinuteSecond Property	3-45
	PowerOnArrival.GPSCoordinates Property	3-45
	PowerOnArrival.Hostname Property	3-45
	PowerOnArrival.Location Property	3-46
	PowerOnArrival.MeassuredPower Property	3-46
	PowerOnArrival.ReferenceLevel Property	3-46
	PowerOnArrival.SourceFrequency Property	3-46
	PowerOnArrival.SourceMode Property	3-46

Table of Contents (Continued)

PowerOnArrival.StartFrequency Property	3-47
PowerOnArrival.StopFrequency Property	3-47
PowerOnArrival.Trace Property	3-47
Methods	3-48
PowerOnArrival.VerifySource Method	3-48
Structures	3-48
POA_Result Structure	3-48
Fields	3-48
Latitude Field	3-48
Longitude Field	3-48
Text Field	3-48
3-10 RSM Class	3-49
Properties	3-49
RSM.AntennaPort Property	3-49
RSM.Attenuation Property	3-49
RSM.AudioMute Property	3-49
RSM.AverageCount Property	3-49
RSM.BWRatio Property	3-49
RSM.CenterFrequency Property	3-50
RSM.ChannelPower Property	3-50
RSM.ConnectionOK Property	3-50
RSM.Detection Property	3-50
RSM.DisplayPoints Property	3-51
RSM.FormatFrequency Property	3-51
RSM.FrequencySpan Property	3-51
RSM.FrequencyValue Property	3-51
RSM.GPS Property	3-52
RSM.GPSDisplay Property	3-52
RSM.IQBandwidth Property	3-52
RSM.IQBitDepth Property	3-53
RSM.IQCaptureLength Property	3-53
RSM.IQ_I Property	3-54
RSM.IQ_Plot Property	3-54
RSM.IQ_Q Property	3-55
RSM.MeasurementStatus Property	3-55
RSM.OBW Property	3-55
RSM.OBWPercent Property	3-55
RSM.Options Property	3-55
RSM.Preamplifier Property	3-56
RSM.ProductID Property	3-56
RSM.RBW Property	3-56
RSM.RBWAuto Property	3-56
RSM.RBWRatio Property	3-56
RSM.ReferenceLevel Property	3-57
RSM.ReferenceLevelOffset Property	3-57
RSM.StartFrequency Property	3-57
RSM.StopFrequency Property	3-57
RSM.Streaming Property (Read Only)	3-58
RSM.SweepMode Property	3-58
RSM.Sweeping Property	3-58
RSM.Trace Property	3-58
RSM.TraceMode Property	3-58
RSM.URL Property	3-58
RSM.Units Property	3-59

Table of Contents (Continued)

RSM.VBW Property	3-59
RSM.VBWAUTO Property	3-59
RSM.VBWType Property	3-59
RSM.Version Property	3-59
RSM.YScale Property	3-60
Methods	3-61
RSM.AMModulation Method	3-61
RSM.CaptureIQ Method	3-61
RSM.CheckOption Method	3-62
RSM.CreatePlayBackWaveFile	3-62
RSM.DeleteFile Method	3-62
RSM.DisplayStoredTrace Method	3-63
RSM.DownloadFile Method	3-63
RSM.GetFiles Method	3-63
RSM.GetFolders Method	3-63
RSM.NewFolder Method	3-63
RSM.PlayIQ_AM Method	3-64
RSM.PlayIQ_FM Method	3-64
RSM.Preset Method	3-64
RSM.Reboot Method	3-64
RSM.RecallMeasurement Method	3-64
RSM.ResetCmdPort Method	3-65
RSM.SaveIQ Method	3-65
RSM.SaveIQ_AM Method	3-65
RSM.SaveIQ_FM Method	3-65
RSM.SaveMeasurement Method	3-66
RSM.SaveSetup Method	3-66
RSM.SendCommand Method	3-66
RSM.SweepGraph Method	3-66
RSM.SweepGraph64 Method	3-66
RSM.TriggerSweep Method	3-67
Events	3-67
OnFrequencyChange Event	3-67
Structures	3-67
FrequencyParameters Structure	3-67
Fields	3-67
CenterFrequency Field	3-67
FrequencySpan Field	3-67
StartFrequency Field	3-68
StopFrequency Field	3-68
Enumerations	3-69
BitDepth Enumeration	3-69
GPSDisplayMode Enumeration	3-69
Structures	3-69
FileHeader Structure	3-69
Fields	3-69
Bandwidth Field	3-69
CaptureLength Field	3-70
CenterFrequency Field	3-70
TimeStamp Field	3-70
Version Field	3-70
3-11 Scanner Class	3-71
Properties	3-71

Table of Contents (Continued)

Scanner.ChannelCount Property	3-71
Scanner.Connected Property	3-71
Scanner.FileDelimiter Property	3-71
Scanner.FileViewer Property	3-71
Scanner.MaxFileSize Property	3-71
Scanner.MinimumFreeSpace Property	3-71
Scanner.ReloadCPC Property	3-72
Scanner.SaveGPS Property	3-72
Scanner.SaveOnEvent Property	3-72
Scanner.SaveTraces Property	3-72
Scanner.ScanDelay Property	3-72
Scanner.ScanOnStartup Property	3-72
Scanner.SetOutputFolder Property	3-72
Scanner.ShowTraces Property	3-73
Scanner.SweepInterval Property	3-73
Methods	3-73
Scanner.AddChannel Method	3-73
Scanner.AddChannelsDialog Method	3-73
Scanner.Close Method	3-73
Scanner.ListChannels Method	3-73
Scanner.LoadConfiguration Method	3-74
Scanner.Pause Method	3-74
Scanner.RemoveAllChannels Method	3-74
Scanner.RemoveChannel Method	3-74
Scanner.RunCounters Method	3-74
Scanner.RunSummary Method	3-74
Scanner.SaveConfiguration Method	3-75
Scanner.Start Method	3-75
Events	3-75
ScanEvent Event	3-75
Structures	3-75
MaskDef Structure	3-75
ScannerChannelDefinition Structure	3-75
Fields	3-76
Level Field	3-76
StartFreq Field	3-76
StopFreq Field	3-76
Antenna Field	3-76
ChannelPower Field	3-76
DeltaMarker Field	3-76
DeltaPower Field	3-77
Description Field	3-77
DisplayPoints Field	3-77
Hostname Field	3-77
ListIndex Field	3-77
LowerLimit Field	3-77
LowerMask Field	3-77
Marker Field	3-78
MarkerPower Field	3-78
OBW Field	3-78
PassRate Field	3-78
PassStatus Field	3-78
Preamp Field	3-78
RBW Field	3-78

Table of Contents (Continued)

RefLevel Field	3-79
StartFreq Field	3-79
StopFreq Field	3-79
TraceCount Field	3-79
UpperLimit Field	3-79
UpperMask Field	3-79
VBW Field	3-79
strTrace Field	3-80
Enumerations	3-81
Delimiters Enumeration	3-81
SaveModes Enumeration	3-81
SaveTrigger Enumeration	3-81
ViewModes Enumeration	3-81
3-12 TDOAControl Class	3-83
Properties	3-83
TDOAControl.AntennaChannel Property	3-83
TDOAControl.AutoReferenceLevels Property	3-83
TDOAControl.AverageCount Property	3-83
TDOAControl.AverageDistances Property	3-83
TDOAControl.CaptureBandwidth Property	3-83
TDOAControl.CenterFrequency Property	3-84
TDOAControl.DrawEstimateLines Property	3-84
TDOAControl.DrawHeatMap Property	3-84
TDOAControl.DrawSweepTrace Property	3-84
TDOAControl.GraphMode Property	3-84
TDOAControl.Latitude Property	3-84
TDOAControl.Longitude Property	3-84
TDOAControl.MovingAverageCount Property	3-85
TDOAControl.Preamp Property	3-85
TDOAControl.ProbeDelay Property	3-85
TDOAControl.ProbeHostname Property	3-85
TDOAControl.ProbeStatus Property	3-85
TDOAControl.ReferenceLevel Property	3-86
TDOAControl.ReportFolder Property	3-86
TDOAControl.RunCount Property	3-86
TDOAControl.ShowCorrelationResults Property	3-86
TDOAControl.ShowGraphs Property	3-86
TDOAControl.ShowHelp Property	3-86
TDOAControl.ShowMap Property	3-87
TDOAControl.ShowMobileTracking Property	3-87
TDOAControl.ShowViewOptions Property	3-87
TDOAControl.TrackMovingTarget Property	3-87
TDOAControl.WaitForPeak Property	3-87
Methods	3-87
TDOAControl.GenerateReport Method	3-87
TDOAControl.GetParametersFromInput Method	3-88
TDOAControl.ResetAveraging Method	3-88
TDOAControl.RunTDOA Method	3-88
TDOAControl.SaveIQData Method	3-88
TDOAControl.ScreenShot Method	3-88
TDOAControl.VerifyProbe Method	3-88
Events	3-89
CalibrateProbes Event	3-89

Table of Contents (Continued)

TDOA_Iteration Event.	3-89
Structures	3-89
ChannelInfo Structure.	3-89
Fields	3-89
Channel Field	3-89
Latitude Field	3-89
Longitude Field	3-89
ProbeName Field	3-90
Ready Field.	3-90
URL Field	3-90
Enumerations.	3-90
ChartType Enumeration	3-90
3-13 Trace Class.	3-90
Trace.StartFrequency().	3-90
Trace.StopFrequency().	3-91
Trace.RBW().	3-91
Trace.dBDown().	3-91
Trace.TraceString().	3-91
Trace.Power(int Index).	3-92
Trace.SortChannels(channelSort Sortby)	3-92
Trace.Channels()	3-92

Chapter 1 — Overview

1-1 Introduction

This documentation provides a description of the Anritsu's remote spectrum analyzer used with Vision Software Application Programming Interface (API). It describes the graphical user interface, provides a how-to chapter to set up measurements, discusses API programming, and covers a lists of API programming commands. API's are presented both for control of Anritsu's remote spectrum analyzer hardware as well as for Vision software. Most features in Vision are paid options. In order for these options to work, the remote spectrum analyzer hardware must have these options enabled. See the Vision software Product Brochure and User Guide for option information.

Additional Documentation

This Vision User Guide is part of a set of manuals that describe all of the remote spectrum analyzer functions and their use with Vision. Instrument operations of various document types are listed [Table 1-1, "Additional Documentation"](#).

Table 1-1. Additional Documentation

Document Part Number	Description
10580-00435	MX280001A Vision Programming Manual
10100-00064	MS2710xA Important Product Information, Compliance, and Safety Guide
10580-00419	MS2710xA Remote Spectrum Monitor User Guide
10580-00420	MS2710xA Maintenance Manual
11410-00898	MS27100A Spectrum Monitor Module Technical Data Sheet
11410-00853	MS27101A Indoor Remote Spectrum Monitor Technical Data Sheet
11410-00847	MS27102A Outdoor Remote Spectrum Monitor Technical Data Sheet
11410-00854	MS27103A Remote Spectrum Monitor Technical Data Sheet
10100-00069	MS2090A Important Product Information, Compliance, and Safety Guide
10580-00444	MS2090A Field Master Pro, Spectrum Analyzer User Guide
11410-01000	MS2090A Field Master Pro Technical Data Sheet
10580-00428	SpectraVision User Guide

The Vision software platform works with the receivers with their options shown in [Table 1-2](#) to automate the process of collecting measurement data.

Table 1-2. Instrument and Vision Enabled Options

Model	Option/Description
MS2710xA	MS2710xA-0400 Vision Monitor Enabled MS2710xA-0401 Vision Locate Enabled (requires Option 400 above) MS2710xA-0407 Vision High-Speed Port Scanner Enabled MS2710xA-0479 Vision AM Demodulation/FM Deviation MS2710xA-0485 Vision Spectrum Occupancy (requires Option 400) MS2710xA-0486 Vision Coverage Mapping (requires Option 407)
MS2090A	(400) Vision Monitor Enabled (407) High-Speed Port Scanner Enabled

Document Conventions

The following conventions are used throughout the Vision MX280001A documentation set.

Terms

The following terms used to identify a target receiver are used interchangeably throughout this programming manual.

(RSM, remote spectrum monitor, remote spectrum analyzer, probe, remote probe, and target receiver)

Dialogs

The Vision dialogs examples displayed in this User Guide are common for all Anritsu target receiver applications.

1-2 System Requirements

Listed below are the minimum requirements for running the Vision API:

- Quad Core Processor – 2 GHz or faster clock speed
- Microsoft Windows 7 (Service Pack 1) or later.
- 8 GB of RAM
- 100 GB of hard-disk space – 1 TB recommended
- Microsoft.Net Framework 4.0 or higher
- LAN connection to all remote sensors
- Vision must be installed on a Windows 64-bit OS

1-3 Using Vision API

The API provides the programmer with a library of building blocks to develop a Vision application program. These building blocks can create the programs for geo-locating RF emitter signals:

- Power On Arrival (POA)
- Time Difference of Arrival (TDOA)

Other programs that can be developed are:

- High-Speed Port Scanner
- Radio Monitoring
- Map Creation. Additionally

The API allows system integrators and programmers to develop specific remote spectrum analyzing and management applications efficiently.

Frequency Settings

When using the Vision programs and setting frequency parameters, the maximum frequency setting Vision provides is the maximum allowed on the measuring instrument up to 54 GHz. Vision does not verify the maximum input frequency range of the measuring instrument.

1-4 An Example Using the Vision API

1. Create a new project in Visual Studio.
 - a. The project shown below is a Console Application written in C#.

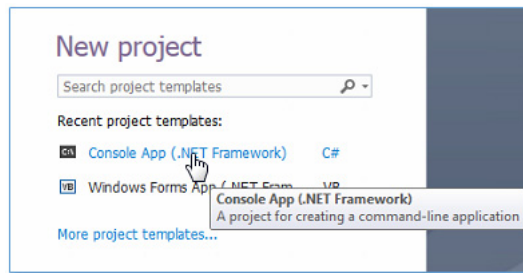


Figure 1-1.

2. Add a reference to Vision API.
 - a. From the Project menu select the **Add Reference...** option.
 - b. Select **Browse** and then locate to add the VisionAPI.dll. This will be found in the Vision Application folder. If you used default installation options, locate C:\Program Files (x86)\Anritsu Company\Vision Remote Monitoring
 - c. In Solution Explorer, verify that the VisionAPI is listed under References for your project.

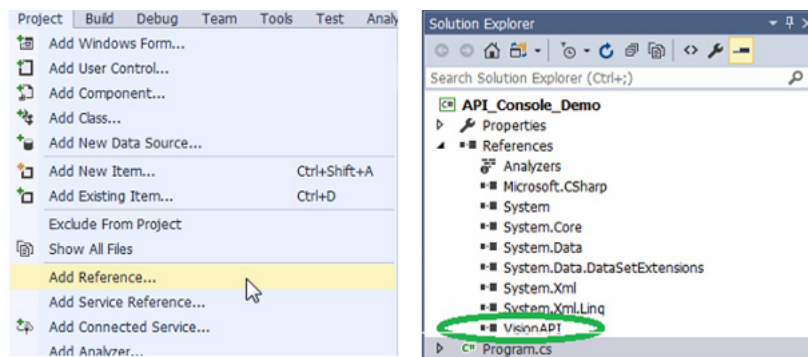


Figure 1-2.

3. Add code to access the VisionAPI functions. (You can cut and paste the code below to try the example shown.)
 - a. Add 'using VisionAPI' to simplify references.
 - b. Declare an object to reference the API library you want to use. In the code example below, [Section "Example - Vision API Console Demo"](#), this is done with 'RSM probe = new RSM();'
 - c. Refer to [Figure 1-3](#) below. Add the files highlighted in blue with NMath.dll for NMath to execute properly.

Name	Date modified	Type	Size
 VisionAPI.dll	6/29/2017 12:46 PM	Application extens...	501 KB
 TDOADemo	6/29/2017 12:55 PM	XML Document	1 KB
 TDOADemo.pdb	6/29/2017 12:55 PM	Program Debug D...	30 KB
 TDOADemo.exe.config	5/2/2017 4:29 PM	XML Configuratio...	2 KB
 TDOADemo	6/29/2017 12:55 PM	Application	17 KB
 NMath.dll	5/2/2017 4:29 PM	Application extens...	1,775 KB
 Microsoft.Solver.Foundation.dll	4/19/2017 6:57 PM	Application extens...	2,546 KB
 gTrackBar.dll	5/2/2017 4:29 PM	Application extens...	68 KB
 x86	7/26/2017 3:24 PM	File folder	

Figure 1-3. NMath.zip files

- d. Use the new object to access properties and methods in the API. In the example code below:
- Frequency parameters are read from the remote spectrum monitor and displayed.
 - A new center frequency is read from the command line and sent to the target receiver.
 - The new value is displayed to show that the target receiver has been set as directed.

Example - Vision API Console Demo

```
using System;
using VisionAPI;

namespace API_Console_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            RSM probe = new RSM();

            Console.WriteLine("Enter RSM URL: ");
            probe.URL = Console.ReadLine();

            Console.WriteLine("Start frequency:  " + probe.StartFrequency);
            Console.WriteLine("Stop frequency:  " + probe.StopFrequency);
            Console.WriteLine("Center frequency: " + probe.CenterFrequency);
            Console.WriteLine("Frequency span:  " + probe.FrequencySpan);
        }
    }
}
```

```
        Console.WriteLine("\nEnter new center frequency: ");  
        probe.CenterFrequency = Console.ReadLine();  
        Console.WriteLine("Center frequency set to: " + probe.CenterFrequency);  
        Console.ReadKey();  
    }  
}
```

4. Compile and run the application. The example is a console application. Below is shown the screen when this is executed.

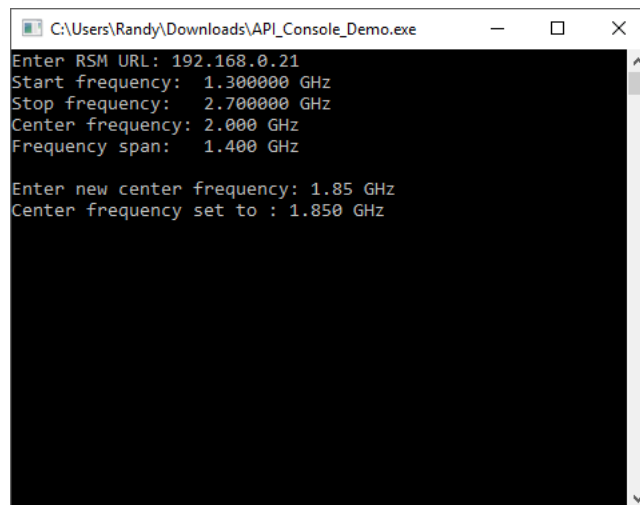


Figure 1-4. Vision API Console Demo

1-5 Vision API User Controls Examples

Three Visual Studio example projects are installed on your PC during the installation of Vision. With the default installation options these three projects are located in C:\Program Files (x86)\Anritsu Company\Vision Remote Monitoring\Vision API Examples.

These example projects can be opened with Visual Studio 2010 or later, including express versions of Visual Studio.

Chapter 2 — Vision User Controls

2-1 Introduction

This section covers the Vision user interface controls.

2-2 File Viewer

The File Viewer component provides a quick means to include trace displays in a host program. File Viewer supports four modes of operation – Trace Array, Local File System, Remote File System, and Channel Power Measurement (.cpm) Files.

Trace Array

When using Trace Array mode, the side panel will normally be hidden, and possibly the spectrogram as well. Trace Array mode allows for a simple trace viewer without file system navigation. If you leave the side panel visible, the user can select a different file mode and change what is shown in the trace window. Trace Array mode is typically used when the trace source is not a standard spectrum monitor or Vision files. For example, traces can be captured using the SCPI interface to the remote spectrum monitor and displayed without using either the RSM file format or the CPM file format.

To use Trace Array mode, hide the side panel and spectrogram panels and call the SetTraceData method. You will need to supply the start and stop frequencies, the desired reference level, and optionally timestamp and GPS coordinates for each trace.

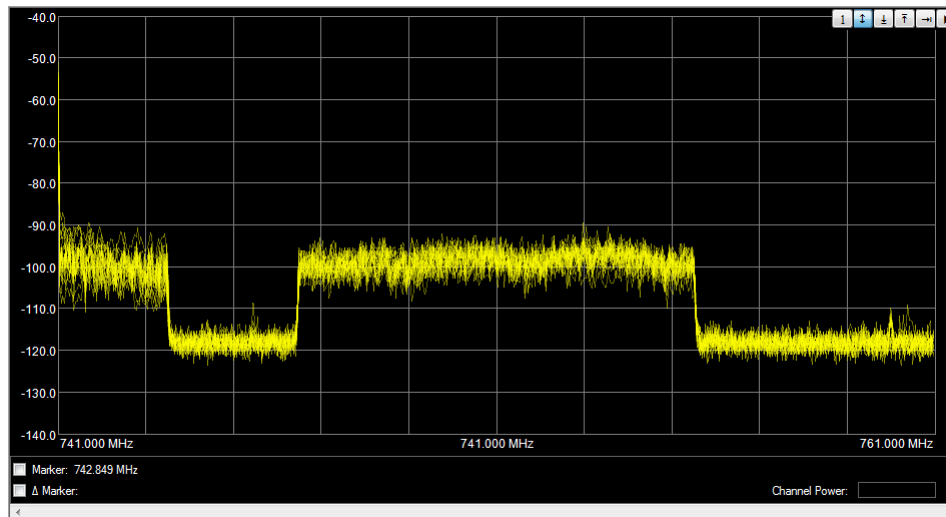


Figure 2-1. Trace Array Mode

Local File System

Local file system allows the user to browse for spectrum monitor files already downloaded and stored on the local file system. You can lock the user into a particular path on the local file system by hiding the side panel. If the side panel is not visible, the application will have to provide an alternative means for setting the filename(s) to be displayed.

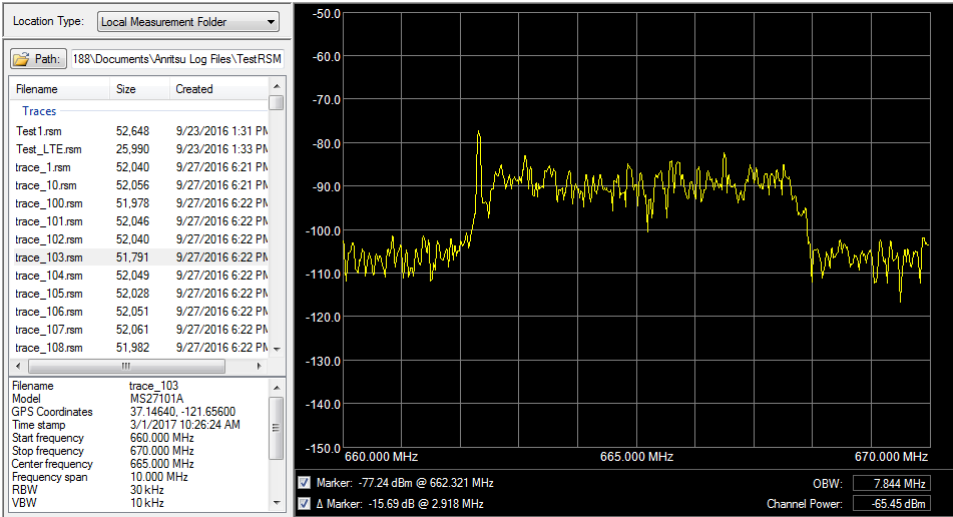


Figure 2-2. Local File System

Remote File System

Remote File system provides a simple means for viewing and downloading measurement files stored on a remote spectrum monitor. You can make the side panel visible, which allows users to browse directly for different target receivers. Alternatively, you can hide the side panel, while providing an alternate means in the host application to select the Hostname.

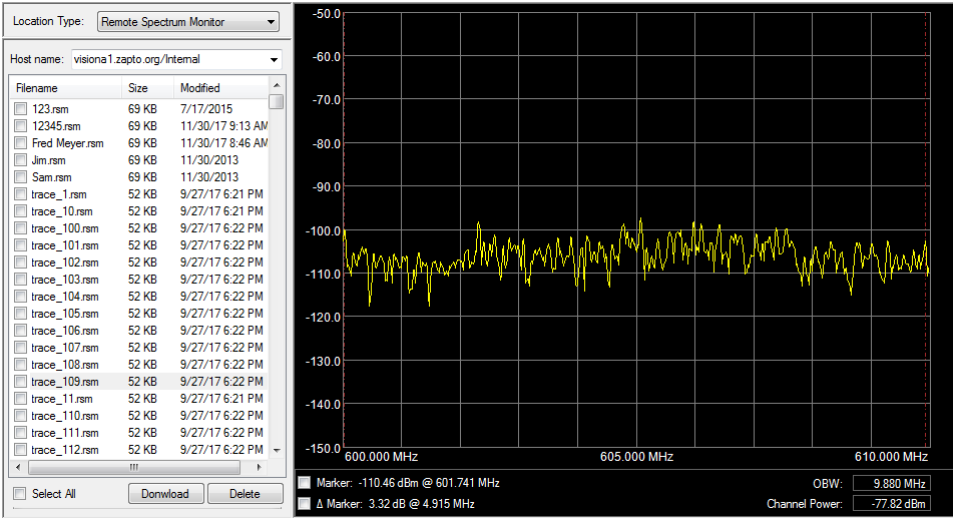


Figure 2-3. Remote File System

Channel Power Measurement (*.cpm) files.

Channel Power Measurement files are created by the Vision High-Speed Channel Scanner API. You can make the side panel visible, allowing users to browse directly for CPM files, or hide the side panel and provide an alternate means in the host application to select a file.

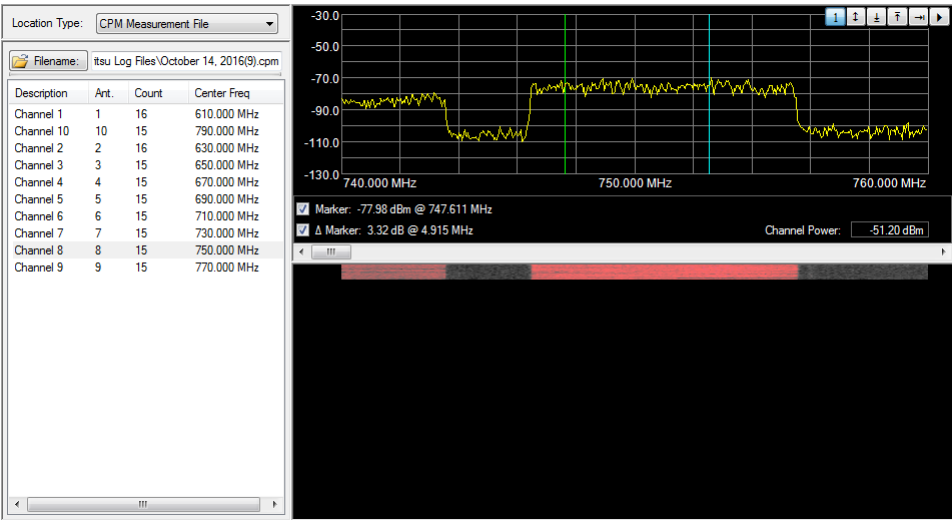


Figure 2-4. Channel Power Measurement Files

2-3 TDOA Panel

The TDOA user control provides a quick means to include TDOA measurements in a host program.

The TDOA user control can be configured several ways to provide the right level of detail and simplicity needed in a host application. Many of the display components can be hidden to provide a basic interface with few options and simple execution. If more control is sought, then various pieces can be added to the UI independently to provide just what is wanted.

Several display combinations are shown. Not all combinations are included, but all of the elements are shown in some configuration.

All optional UI components can be made visible or hidden at design-time by setting properties in the visual editor. There is a View button in the button bar that can be hidden or visible. If this button is visible, then users can control which components they have access to at run-time.

Basic User Interface (UI)

The basic UI provides a minimum set of control to do a TDOA measurement. This includes setting the Hostname for the probes to be used, along with the center frequency and bandwidth of the target signal. You also can turn on/off the preamps in each probe and optionally set the reference level, or have Vision TDOA automatically set the reference level for you. In the example below, example host names are used for each of the 3 probes. Use your own URL or IP address in your application.

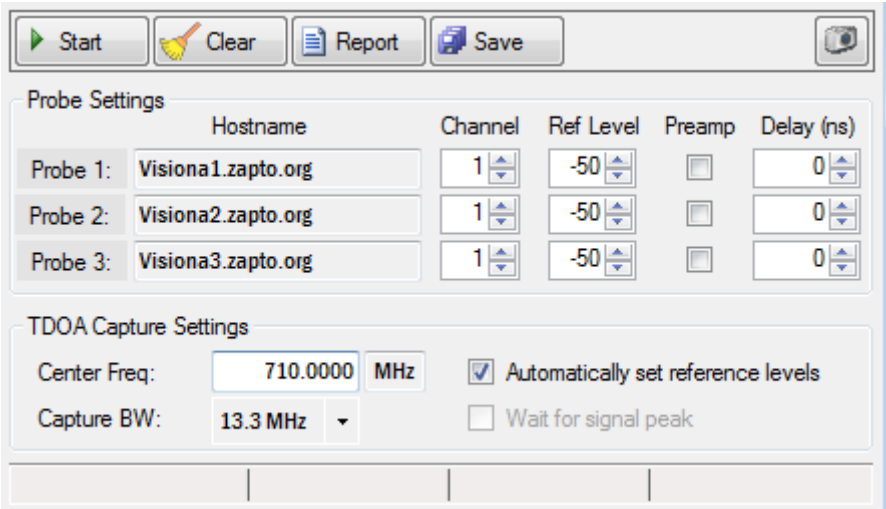


Figure 2-5. Basic User Interface

Correlation Results and Map

Correlation results are not necessary and many users new to TDOA will not know how to interpret them. These results can be hidden to simplify the interface.

In most cases, showing the map will be desirable. The map is crucial to visualizing the TDOA output. However, you can hide the map display.

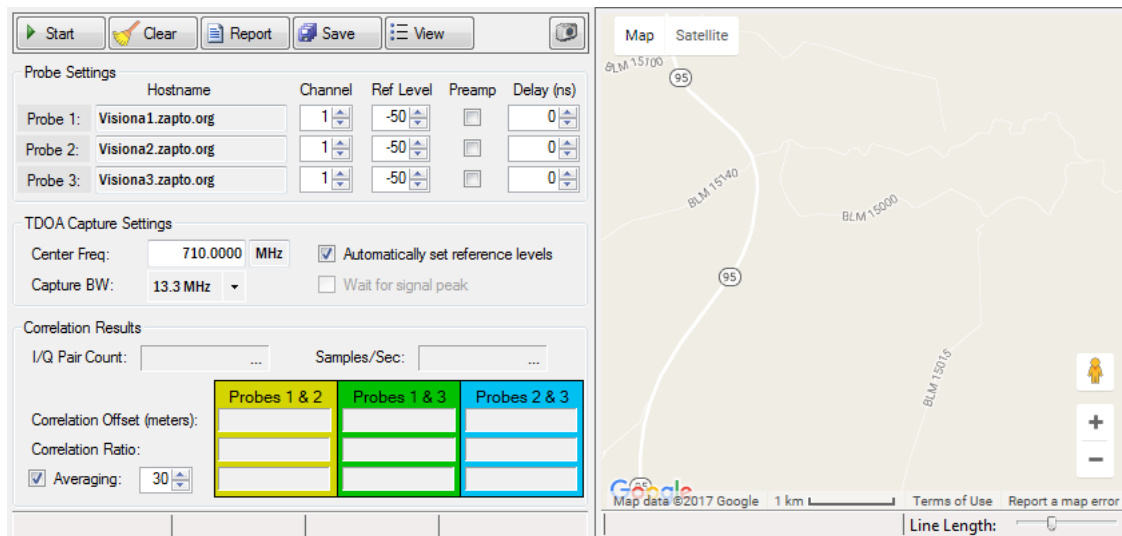


Figure 2-6. Correlation Results and Map

Correlation Results and IQ Graphs

I/Q graphs can be shown. The I/Q diagram is very useful for interpreting the validity of the data captures. The graphs, when visible, can contain either the raw I/Q data or show a trace sweep for each I/Q data capture.

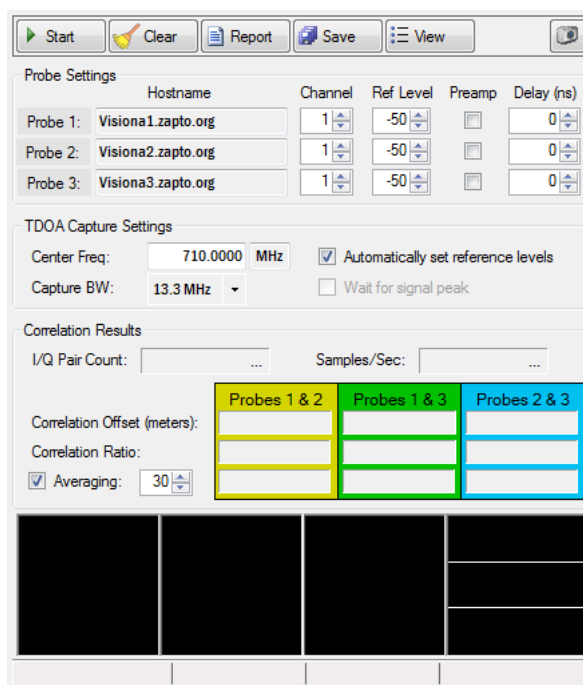


Figure 2-7. Correlation Results and IQ Graphs

Moving Target Tracking

In the image below, the Tracking option is enabled. With Tracking enabled, results are repeatedly calculated and added to a table after the run is complete. The results in the table can be exported to a KML file. KML files are useful for importing tracks to mapping programs.

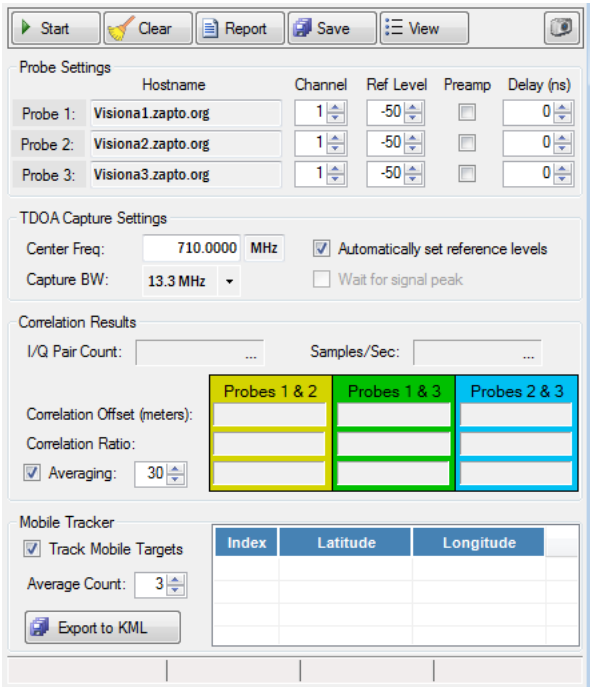


Figure 2-8. Tracking Option Enabled

Full User Interface

All optional components enabled. The View button in the button bar can be disabled (as shown in the Basic UI below). When this is enabled, users can turn on each UI component as they wish. To truly restrict users to a predefined configuration, the View button must be hidden in the development stage.

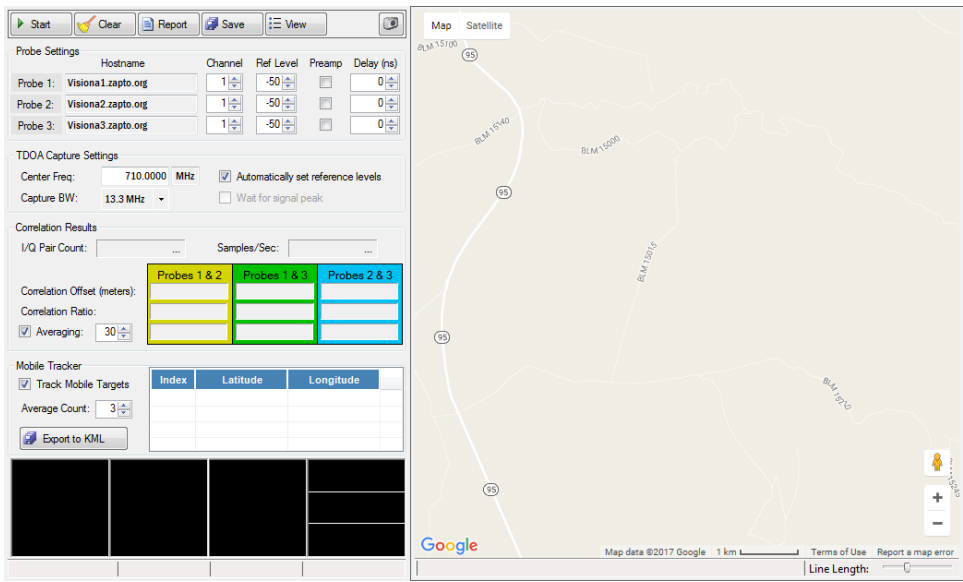


Figure 2-9. Disabled View Button

2-4 Radio Monitor

The Radio Monitor user control combines several functions related to I/Q capture and AM/FM demodulation into a single interface.

The Radio monitor has four graph regions for displaying captured trace and I/Q data.

Lower-Left hand corner is the I/Q plot. There is a speaker icon on this plot that will attempt to demodulate and play a sound file from the captured I/Q data. This will just be noise if the capture is off-peak, or is too wide and contains a lot of noise or a mixture of other signals.

Top right is the RF sweep trace plot. This is a density plot, so traces are added to it as they are collected. The density plot is reset each time you press the RF Sweep button to begin a new acquisition.

Middle right is the Audio file plot. It displays the audio output from the demodulation. There is also a Speaker Icon on this window to replay what is shown in this window.

At the bottom right of the Remote Monitor window is the Spectrogram plot. This shows the same traces as the RF sweep graph, but in a spectrogram view. You can often tell that certain frequencies are FM modulated by the way the peak wiggles back and forth in the spectrogram.

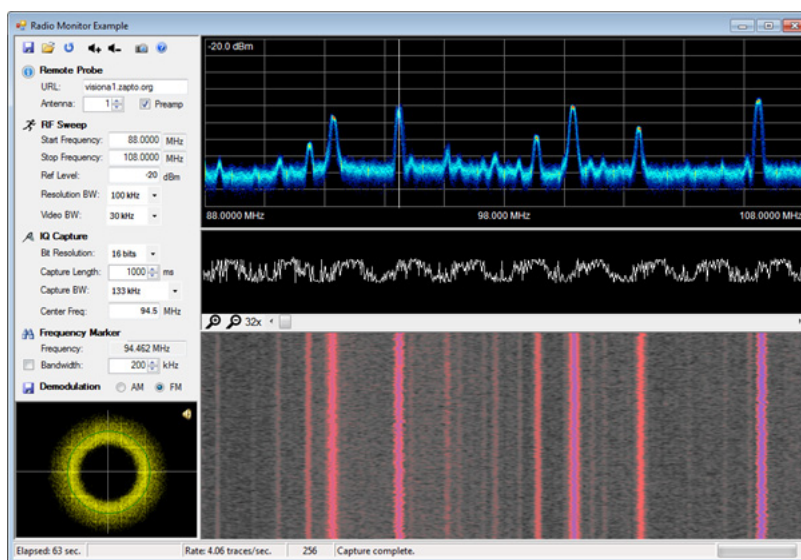


Figure 2-10. Radio Monitor Example

Radio Monitor Icons

At the left of the Radio Monitor window are the main menu icons and program buttons that are defined below.

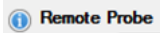
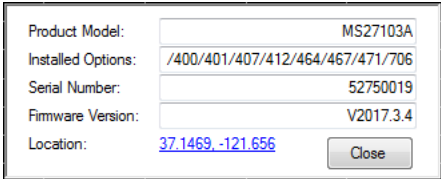
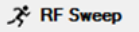
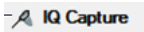
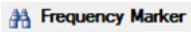
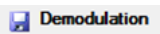


Figure 2-11. Radio Monitor Main Menu Icons

Table 2-1. Radio Monitor Icon Definition

Save	Saves the current I/Q capture data to a disk file.
Open	Loads a saved I/Q Capture file into the Radio Monitor program. A file may take a few seconds to several minutes to load. For wide bandwidth capture files, the file is split into time slices and FFT's are used to get back the RF sweep for each time slice. Once the file is open, the trace data will be displayed and selecting a frequency in the trace graph will allow you to listen to an AM or FM signal at any frequency within the larger I/Q capture.
Refresh	Reloads the file header for the currently opened I/Q Capture file, if one exists. This button resets the parameter fields to match the loaded I/Q Capture file.
Volume Up	Increases the volume.
Volume Down	Decreases the volume.
Screenshot	Takes a picture of the Radio Monitor control and copies it to the Windows Clipboard.
Help	Opens online Help. Only visible if a file is assigned using the HelpFile property.

Radio Monitor Buttons

Button	Description
	Remote Probe: The Remote Probe button captures basic system information for the designated probe. 
	RF Sweep: The RF Sweep button starts collecting traces from the target receiver and draws them in a density plot. The Sweep parameters specified in the section under this button are first sent to setup the sweep measurements.
	I/Q Capture: The I/Q Capture button captures I/Q data from the target receiver. The bit resolution, capture length, bandwidth and center frequency are set according to the input fields below this button.
	Frequency Marker: The Frequency Marker button does a peak search in the RF Sweep window and sets a marker at the peak power position. The Frequency is shown in the box below this button. The marker can also show a width that can be used to determine the appropriate I/Q capture bandwidth for a peak of interest.
	Demodulation: The Save Demodulation button causes the most recently played AM/FM demodulation sound to be saved to a WAV file. Click the AM or FM radio buttons to select the demodulation mode.

Chapter 3 — Vision API Reference

3-1 Introduction

This Vision API supports the following programming classes:

- “Acquire Class”
- “AngleOfArrival Class”
- “Database Class”
- “FileViewer Class”
- “MaskEditor Class”
- “Monitor Class”
- “Occupancy Class”
- “PowerOnArrival Class”
- “RSM Class”
- “Scanner Class”
- “TDOAControl Class”
- “Trace Class”

3-2 Acquire Class

The Acquire Class provides programmatic access to all of the features of Vision Acquire. Each of the properties and methods described below correspond to a function or feature in Vision acquire accessed through the Vision Acquire UI. The descriptions below are meant to show how to access these functions. Refer to the Vision User Guide.

Properties

Acquire.ArchivePeriod Property

Gets or sets the data collection period.

```
VB: Public Property ArchivePeriod() As ArchivePeriods
C#: public ArchivePeriods ArchivePeriod {get; set;}
[C++]: public ArchivePeriods ArchivePeriod {get; set;}
```

Acquire.AutoArchive Property

Automatically archive the trace tables into the database.

```
VB: Public Property AutoArchive() As Boolean
C#: public bool AutoArchive {get; set;}
C++: public bool AutoArchive {get; set;}
```

Remarks

Next Archive will notify you when the next archiving process will take place.

Acquire.AutoClearOld Property

Removes old traces from the database.

```
VB: Public Property AutoClearOld() As Boolean
C#: public bool AutoClearOld {get; set;}
C++: public bool AutoClearOld {get; set;}
```

Remarks

Old traces are removed once each day following the first data collection after 6:00 am.

Acquire.Connected Property

Gets the connection status of the remote spectrum monitor.

```
VB: Public ReadOnly Property Connected() As Boolean
C#: public bool Connected {get;}
C++: public bool Connected {get;}
```

Acquire.Database Property

Obtains the name of the database in the active remote spectrum monitor.

```
VB: Public ReadOnly Property Database() As String
C#: public string Database {get;}
C++: public string Database {get;}
```

Acquire.GetNotification Property

A Notification will be sent to email set in Parameters.

```
VB: Public ReadOnly Property GetNotification(ByVal Email As String) As NotifyType
C#: [System.Runtime.CompilerServices.IndexerName("GetNotification")]
    public NotifyType this[string Email] {get;}
C++: [System.Runtime.CompilerServices.IndexerName("GetNotification")]
    public NotifyType this[string Email] {get;}
```

Parameters: Email: The email address of the person or account to be notified.

Acquire.GetNotificationList Property

Obtains the list of emails to be sent Notifications.

```
VB: Public ReadOnly Property GetNotificationList() As List(Of String)
C#: public List<string> GetNotificationList {get;}
C++: public List<string> GetNotificationList {get;}
```

Acquire.IsRunning Property

Returns a Boolean value indicating True if Vision Acquire is currently running on the same PC; or False if Vision Acquire is not running. This does not indicate the sweeping state, just whether or not the program is running. If True, then you can safely use other Acquire properties and methods to interact with Vision Acquire.

```
VB: Public ReadOnly Property IsRunning() As Boolean
C#: public bool IsRunning {get;}
C++: public bool IsRunning {get;}
```

Acquire.NotificationCount Property

Gets the number of notifications in the notification list.

VB: Public ReadOnly Property NotificationCount() As [Integer](#)

C#: public [int](#) NotificationCount {get;}

C++: public [int](#) NotificationCount {get;}

Acquire.SendNotifications Property

Gets the status of send notifications or sets the command to send notification.

VB: Public Property SendNotifications() As [Boolean](#)

C#: public [bool](#) SendNotifications {get; set;}

C++: public [bool](#) SendNotifications {get; set;}

Acquire.SweepInterval Property

Gets the current sweep interval or sets a new sweep interval.

VB: Public Property SweepInterval() As [String](#)

C#: public [string](#) SweepInterval {get; set;}

C++: public [string](#) SweepInterval {get; set;}

Acquire.SweepState Property

Gets or sets the Sweep State.

VB: Public Property SweepState() As [Boolean](#)

C#: public [bool](#) SweepState {get; set;}

C++: public [bool](#) SweepState {get; set;}

Acquire.ThreadCount Property

Gets the current thread count or sets a new thread count.

VB: Public Property ThreadCount() As [Integer](#)

C#: public [int](#) ThreadCount {get; set;}

C++: public [int](#) ThreadCount {get; set;}

Remarks

The number of threads represents the number of probes under simultaneous communication. It cannot be more than the number of probes in the database.

Methods

Acquire.AddChannel2Probe Method

Adds new channels to the list according to the parameters.

VB: Public Sub AddChannel2Probe(ByVal *Probename* As String,ByVal *Channel* As ChannelInfo)

C#: public System.Void AddChannel2Probe(string *Probename*,ChannelInfo *Channel*)

C++: public System.Void AddChannel2Probe(string *Probename*,ChannelInfo *Channel*)

Parameters: Probename: The name of the probe where the channel is to be added.

Channel: A structure of type ChannelInfo that holds the description of the channel to add.

Acquire.AddGroup Method

Creates a group with the name set in parameters. Group names are used to categorize remote probes. The group name should be a descriptive name and apply to a sub-set of remote spectrum analyzers that you want to see as a set. A group name might correspond to a geographic area, the name of a technician responsible for oversight, or to a transmission or modulation type.

VB: Public Sub AddGroup(ByVal *GroupName* As String)

C#: public System.Void AddGroup(string *GroupName*)

C++: public System.Void AddGroup(string *GroupName*)

Parameters: GroupName: The name of a group to add.

Acquire.AddNotification Method

Adds a new notification with the parameters Email and Mode.

VB: Public Sub AddNotification(ByVal *Email* As String,ByVal *Mode* As NotifyType)

C#: public System.Void AddNotification(string *Email*,NotifyType *Mode*)

C++: public System.Void AddNotification(string *Email*,NotifyType *Mode*)

Parameters: Email: The email address of the person or account to notify.

Mode: The notification mode of type NotifyType. This indicates how often this account receives notifications.

Acquire.AddProbe2Database Method

Enters a new probe into the database with the parameter Probe.

VB: Public Sub AddProbe2Database(ByVal *Probe* As **ProbeInfo**)

C#: public **System.Void** AddProbe2Database(**ProbeInfo** *Probe*)

C++: public **System.Void** AddProbe2Database(**ProbeInfo** *Probe*)

Parameters: Probe: Probe description contained in a ProbeInfo structure

Acquire.Archive Method

Saves the current database to a different directory location.

VB: Public Sub Archive()

C#: public **System.Void** Archive()

C++: public **System.Void** Archive()

Acquire.ClearNotifications Method

Disables the send email notifications function.

VB: Public Sub ClearNotifications()

C#: public **System.Void** ClearNotifications()

C++: public **System.Void** ClearNotifications()

Acquire.Compress Method

Immediately compresses the data in the database file.

VB: Public Sub Compress()

C#: public **System.Void** Compress()

C++: public **System.Void** Compress()

Remarks

It removes the unused space created by the deletion of old traces from the database. If Vision software seems to be running unusually slow, compacting the database may be necessary.

Acquire.ConfigureEmail Method

Configure email notification with the Parameters set.

VB: Public Sub ConfigureEmail(*ByVal Hostname* As [String](#),*ByVal PortNumber* As [String](#),*ByVal Authenticate* As [Boolean](#),*ByVal UserName* As [String](#),*ByVal Password* As [String](#))

C#: public [System.Void](#) ConfigureEmail([string](#) *Hostname*,[string](#) *PortNumber*,[bool](#) *Authenticate*,[string](#) *UserName*,[string](#) *Password*)

C++: public [System.Void](#) ConfigureEmail([string](#) *Hostname*,[string](#) *PortNumber*,[bool](#) *Authenticate*,[string](#) *UserName*,[string](#) *Password*)

Parameters: **Hostname:** Hostname is the outgoing server name, or URL. You will need an account on this server to send mail.

PortNumber: The port number on the mail server. This number should be provided by the mail manager.

Authenticate: Specifies if authentication is required (true), or if not (false).

UserName: The username for the registered account on the mail server that you will be using to send mail.

Password: The password corresponding to the user account you are using to send mail.

Acquire.CreateDatabase Method

Sets the directory path for the base station definition files and database files.

VB: Public Sub CreateDatabase(*ByVal Path* As [String](#))

C#: public [System.Void](#) CreateDatabase([string](#) *Path*)

C++: public [System.Void](#) CreateDatabase([string](#) *Path*)

Parameters: **Path:** The full path to a directory where you would like to create a new database. This is a path name and does not include a file name. If a Vision database already exists at this location, it will be overwritten.

Acquire.DeleteNotification Method

Deletes the email address set in parameters.

VB: Public Sub DeleteNotification(*ByVal Email* As [String](#))

C#: public [System.Void](#) DeleteNotification([string](#) *Email*)

C++: public [System.Void](#) DeleteNotification([string](#) *Email*)

Parameters: **Email:** The email address of the person or account to be modified.

Acquire.LoadDatabase Method

Loads the contents of a base station definition file (.def) set in parameters into the Vision database.

VB: Public Sub LoadDatabase(*ByVal Databasename* As [String](#))

C#: public [System.Void](#) LoadDatabase([string Databasename](#))

C++: public [System.Void](#) LoadDatabase([string Databasename](#))

Parameters: Databasename: The full path to a directory where you would like to create a new database. This is a path name and does not include a file name. If a Vision database already exists at this location, it will be overwritten.

Acquire.ModifyChannel Method

Modify a channel set in Parameters.

VB: Public Sub ModifyChannel(*ByVal Probename* As [String](#),*ByVal Port* As [Integer](#),*ByVal Channel* As [ChannelInfo](#))

C#: public [System.Void](#) ModifyChannel([string Probename](#),[int Port](#),[ChannelInfo Channel](#))

C++: public [System.Void](#) ModifyChannel([string Probename](#),[int Port](#),[ChannelInfo Channel](#))

Parameters: Probename: The name of the probe to be modified.

Port: The port number or antenna channel to modify. Ports or channel typically refer to the antenna port on a multi-port target receiver files.

Channel: Channel description loaded into a ChannelInfo structure.

Acquire.ModifyProbeDefinition Method

Select and modify the probe settings with Parameters.

VB: Public Sub ModifyProbeDefinition(*ByVal Probename* As [String](#),*ByVal Probe* As [ProbeInfo](#))

C#: public [System.Void](#) ModifyProbeDefinition([string Probename](#),[ProbeInfo Probe](#))

C++: public [System.Void](#) ModifyProbeDefinition([string Probename](#),[ProbeInfo Probe](#))

Parameters: Probename: The name of the probe to be modified.

Probe: The complete new description of the probe as contained in the ProbeInfo structure.

Acquire.RemoveGroup Method

Deletes a remote spectrum monitor Group set by Parameters from the database.

VB: Public Sub RemoveGroup(*ByVal GroupName* As [String](#))

C#: public [System.Void](#) RemoveGroup([string GroupName](#))

C++: public [System.Void](#) RemoveGroup([string GroupName](#))

Parameters: GroupName: The name of the group to remove from the database. Any probes that are assigned to this group will become orphans. They will show up in the probe list in Vision Monitor, but will not be listed under a particular group name.

Acquire.RemoveProbeFromDatabase Method

Deletes a remote spectrum monitor, set in Parameters, from the database.

VB: Public Sub RemoveProbeFromDatabase(ByVal *ProbeName* As String)

C#: public System.Void RemoveProbeFromDatabase(string *ProbeName*)

C++: public System.Void RemoveProbeFromDatabase(string *ProbeName*)

Parameters: *ProbeName*: The name of an target receiver to be deleted from the database. This will delete all trace history. If you want to preserve the trace history, make a copy of the database before removing the probe. You can make a copy by Archiving the database.

Acquire.SetNotification Method

Sets email notification with Parameters.

VB: Public Sub SetNotification(ByVal *Email* As String,ByVal *Mode* As NotifyType)

C#: public System.Void SetNotification(string *Email*,NotifyType *Mode*)

C++: public System.Void SetNotification(string *Email*,NotifyType *Mode*)

Parameters: *Email*: The email address of the person or account to be modified.

Mode: The new notification mode for this email address.

Acquire.StartSweeping Method

Begin collection of traces to the database.

VB: Public Sub StartSweeping()

C#: public System.Void StartSweeping()

C++: public System.Void StartSweeping()

Acquire.StopSweeping Method

End collection of traces to the database.

VB: Public Sub StopSweeping()

C#: public System.Void StopSweeping()

C++: public System.Void StopSweeping()

Acquire.SweepOnce Method

Makes an immediate single sweep of the remote spectrum monitors.

VB: Public Sub SweepOnce()

C#: public System.Void SweepOnce()

C++: public System.Void SweepOnce()

Acquire.TestNotification Method

Sends a test email to the probe set in Parameters. This is used to test the notification set up and configuration. You must have already specified the email server configuration using the Configure Email method.

VB: Public Sub TestNotification(ByVal *Address* As String)

C#: public System.Void TestNotification(string *Address*)

C++: public System.Void TestNotification(string *Address*)

Parameters: *Address*: The email address at which you want to receive a test email from Vision.

Acquire.Trim Method

Removes old traces from the database immediately.

VB: Public Sub Trim()

C#: public [System.Void](#) Trim()

C++: public [System.Void](#) Trim()

Remarks

This function trims the database so there are about 2000 traces per channel. If you have fewer traces in the database, no changes will be made. If Vision software seems to be running unusually slow, compacting the database may be necessary.

Acquire.ViewLog Method

Opens Vision Log Viewer which contains the activity between the Vision Acquire software and the remote spectrum monitors.

VB: Public Sub ViewLog()

C#: public [System.Void](#) ViewLog()

C++: public [System.Void](#) ViewLog()

Remarks

If opened and a sweep is occurring, real-time data can be viewed.

Structures

ChannelInfo Structure

VB: Public Structure ChannelInfo

C#: public struct ChannelInfo

C++: public struct ChannelInfo

Fields

Active Field

Boolean parameter that indicates if the channel is actively scanned by Vision Acquire.

VB: Public Active As [Boolean](#)

C#: public [bool](#) Active

C++: public [bool](#) Active

Remarks

Setting this to true will cause the corresponding channel to sweep and for the trace data to be stored each time Vision Acquire does a pass through all of the channels in the active database. If Scanning is turned off in Vision Acquire, this does not turn it on, so it does not guarantee that the channel will actually be scanned.

Antenna Field

Active antenna port number of multiport remote spectrum monitor.

VB: Public Antenna As [Integer](#)

C#: public [int](#) Antenna

C++: public [int](#) Antenna

ChannelDelay Field

A calibration setting for doing TDOA measurements.

VB: Public ChannelDelay As [Single](#)

C#: public [float](#) ChannelDelay

C++: public [float](#) ChannelDelay

Remarks

This field holds a time in nanoseconds that represents the time it takes from when a signal hits the antenna until it is time stamped internally. The timing is very important for TDOA measurements. The Channel Delay comes from several factors, such as antenna cable length, GPS antenna location relative to the RF antenna and the GPS cable length. It also includes a small contribution due to internal manufacturing or component difference from probe to probe. This value is typically calculated and set by the TDOA calibration routine.

Direction Field

If this is a directional antenna, perhaps a sector on a cell phone tower, for instance, this value indicates the direction angle from true north for the center line of the antenna sector. Due East is at 90 degrees.

VB: Public Direction As [Integer](#)

C#: public [int](#) Direction

C++: public [int](#) Direction

FFT Field

Sets the FFT field for measurement configuration.

VB: Public FFT As [Boolean](#)

C#: public [bool](#) FFT

C++: public [bool](#) FFT

Frequency Field

A frequency for which the channel is purposed for.

VB: Public Frequency As [Single](#)

C#: public [float](#) Frequency

C++: public [float](#) Frequency

Remarks

This value does not affect the sweep parameters of the remote probe. It is used to verify that two probes are at the same frequency for interference hunting. This is typically the center frequency where the probe is sweeping. A multiport probe might have 12 antenna ports, arranged in sets of three for different sectors monitoring different signals on a single tower. Perhaps monitoring LTE, GSM and WCDMA. When Vision is asked to do certain measurements, such as Power on Arrival, it wants to verify that the channels selected actually are looking at the same frequency range. Someone may adjust the start and stop scan frequencies to see a slightly wider bandwidth, or capture something along one band edge. This frequency field allows a simple way to know if the traces stored in the database are compatible for comparison and estimation.

MaskName Field

The name of the mask used to validate trace data in Vision Acquire. Masks are stored in the database, and the name used here is the name the mask has in the database.

VB: Public MaskName As [String](#)

C#: public [string](#) MaskName

C++: public [string](#) MaskName

Preamp Field

Sets the Preamp field for measurement configuration.

VB: Public Preamp As [Boolean](#)

C#: public [bool](#) Preamp

C++: public [bool](#) Preamp

RBW Field

The Resolution Bandwidth setting for the channel.

VB: Public RBW As [Single](#)

C#: public [float](#) RBW

C++: public [float](#) RBW

RefLevel Field

The Reference Level in dBm for the channel.

VB: Public RefLevel As [Single](#)

C#: public [float](#) RefLevel

C++: public [float](#) RefLevel

RefLevelOffset Field

The Reference Level offset in dB.

VB: Public RefLevelOffset As [Single](#)

C#: public [float](#) RefLevelOffset

C++: public [float](#) RefLevelOffset

StartFreq Field

The sweep Start Frequency in Hz.

```
VB: Public StartFreq As Double
C#: public double StartFreq
C++: public double StartFreq
```

StopFreq Field

The sweep Stop Frequency in Hz.

```
VB: Public StopFreq As Double
C#: public double StopFreq
C++: public double StopFreq
```

SweepTimeOut Field

A value that sets the number of seconds Vision Acquire waits for a probe to respond before moving on.

```
VB: Public SweepTimeOut As Integer
C#: public int SweepTimeOut
C++: public int SweepTimeOut
```

Remarks

Sometime a probe will become unresponsive. We do not want to hold up monitoring all other channels and probes waiting for one to come back on line. How long Vision Acquire should wait is a function of the actual sweep rate, which can vary a lot depending on sweep parameters, and also network or connection latency and throughput.

TraceMode Field

Sets the trace mode: Normal, Averaging, Max Hold, Min Hold.

```
VB: Public TraceMode As Integer
C#: public int TraceMode
C++: public int TraceMode
```

TracePoints Field

The number of data points in each sweep.

```
VB: Public TracePoints As Integer
C#: public int TracePoints
C++: public int TracePoints
```

VBW Field

The Video bandwidth setting for the channel.

```
VB: Public VBW As Single
C#: public float VBW
C++: public float VBW
```

Enumerations

ArchivePeriods Enumeration

ArchivePeriods enumeration is used to set how often Vision Acquire archives a database.

Value:

Table 3-1. ArchivePeriods Values and Definitions

Value	Definition
Hours3	Archive the database every 3 hours.
Hours6	Archive the database every 3 hours.
Hours12	Archive the database every 3 hours.
Days1	Archive the database every day.
Days2	Archive the database every 2 days.
Days4	Archive the database every 4 days.
Days7	Archive the database every 7 days.
Weeks2	Archive the database every 2 weeks.
Weeks4	Archive the database every 4 weeks.
Weeks8	Archive the database every 8 weeks.
Months3	Archive the database every 3 months.
Months6	Archive the database every 6 months.
Year1	Archive the database every year.

NotifyType Enumeration

NotifyType sets the notification type for sending reports.

Value:

Table 3-2. NotifyType Values and Definitions

Value	Definition
None	Do Not send status e-mails to the designated recipient.
Sweep	Send status e-mails to this recipient after every run through the database.
Daily	Send status e-mails to this recipient after every day.
Weekly	Send status e-mails to this recipient once each week.

3-3 AngleOfArrival Class

Properties

AngleOfArrival.AntennaAngle Property

Gets or sets the mid-line direction of the antenna as degrees from true North measured clock-wise.

```
VB: Public Property AntennaAngle(ByVal Index As Integer) As Integer
C#: [System.Runtime.CompilerServices.IndexerName("AntennaAngle")]
    public int this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("AntennaAngle")]
    public int this[int Index] {get; set;}
```

Parameters: Index: Index of the antenna being set.

Remarks

Angle of arrival assumes three antennas on the same tower, each looking at in different directions, or sectors. The sectors are at 120 degree intervals, thus seeing a full 360 degree circle.

AngleOfArrival.DegreeMinuteSecond Property

Gets or sets the format of GPS coordinate.

```
VB: Public Property DegreeMinuteSecond() As Boolean
C#: public bool DegreeMinuteSecond {get; set;}
C++: public bool DegreeMinuteSecond {get; set;}
```

Remarks

GPS coordinates are given in degrees. If AngleOfArrival.DegreeMinuteSecond is True, then the format is: dd° mm' ss.ss" If AngleOfArrival.DegreeMinuteSecond is False, then the format is in decimal, ie. dd.ddddd

AngleOfArrival.GPSCoordinates Property

Does an immediate query to the monitor for updated GPS location.

```
VB: Public Property GPSCoordinates() As PointF
C#: public PointF GPSCoordinates {get; set;}
C++: public PointF GPSCoordinates {get; set;}
```

AngleOfArrival.Hostname Property

Enter the hostname or URL address for the instrument taking measurements.

```
VB: Public Property Hostname() As String
C#: public string Hostname {get; set;}
C++: public string Hostname {get; set;}
```

AngleOfArrival.Location Property

Gets the Power-On-Arrival location estimate as a set of GPS coordinates.

```
VB: Public ReadOnly Property Location() As AOA_Result
C#: public AOA_Result Location {get;}
C++: public AOA_Result Location {get;}
```

Remarks

This Property may return an invalid or useless value if the measurement has not already been properly set up. Before reading the Location, you must set the datasource and the interferer frequency, at least.

AngleOfArrival.MeasuredPower Property

Gets or sets the measured power value set in Parameters.

```
VB: Public Property MeasuredPower(ByVal Index As Integer) As Single
C#: [System.Runtime.CompilerServices.IndexerName("MeasuredPower")]
    public float this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("MeasuredPower")]
    public float this[int Index] {get; set;}
```

Parameters: Index: Index of the antenna being set.

AngleOfArrival.ReferenceLevel Property

Gets or sets the reference level amplitude value.

```
VB: Public Property ReferenceLevel() As Single
C#: public float ReferenceLevel {get; set;}
C++: public float ReferenceLevel {get; set;}
```

AngleOfArrival.SourceFrequency Property

Gets or sets the frequency of the source of interference.

```
VB: Public Property SourceFrequency() As Double
C#: public double SourceFrequency {get; set;}
C++: public double SourceFrequency {get; set;}
```

AngleOfArrival.SourceMode Property

Gets or sets the Source Mode: power at reference frequency, sweep traces from which Vision will extract the power at the source frequency, or live trace mode where Vision will extract the power at the source frequency.

```
VB: Public Property SourceMode() As PowerSource
C#: public PowerSource SourceMode {get; set;}
C++: public PowerSource SourceMode {get; set;}
```

AngleOfArrival.StartFrequency Property

Gets or sets the start frequency for the active remote spectrum monitor.

```
VB: Public Property StartFrequency() As Double
C#: public double StartFrequency {get; set;}
C++: public double StartFrequency {get; set;}
```

AngleOfArrival.StopFrequency Property

Gets or sets Stop Frequency of the active remote spectrum monitor.

VB: Public Property StopFrequency() As [Double](#)
C#: public [double](#) StopFrequency {get; set;}
C++: public [double](#) StopFrequency {get; set;}

AngleOfArrival.Trace Property

Gets or sets the trace of the specified channel set in parameters.

VB: Public Property Trace(ByVal *Index* As [Integer](#)) As [String](#)
C#: [System.Runtime.CompilerServices.IndexerName("Trace")]
public [string](#) this[[int](#) *Index*] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("Trace")]
public [string](#) this[[int](#) *Index*] {get; set;}

Parameters: Index: Index of the antenna being set.

Methods

AngleOfArrival.VerifySource Method

Gets status of Verify Source to set the correct type of angle of arrival calculation depending on the source mode setting.

VB: Public Function VerifySource() As [Boolean](#)
C#: public [bool](#) VerifySource()
C++: public [bool](#) VerifySource()

Structures

AOA_Result Structure

VB: Public Structure AOA_Result
C#: public struct AOA_Result
C++: public struct AOA_Result

AntennaInfo Structure

VB: Public Structure AntennaInfo
C#: public struct AntennaInfo
C++: public struct AntennaInfo

Fields

Angle Field

The angle, measured from North, along which the interferer is located.

VB: Public Angle As [Integer](#)
C#: public [int](#) Angle
C++: public [int](#) Angle

Latitude Field

The latitude coordinate of the remote spectrum monitor.

VB: Public Latitude As [Single](#)

C#: public [float](#) Latitude

C++: public [float](#) Latitude

Longitude Field

The longitude coordinate of the remote spectrum monitor.

VB: Public Longitude As [Single](#)

C#: public [float](#) Longitude

C++: public [float](#) Longitude

Width Field

The width, in degrees, of the location estimation. This can be thought of as a pie wedge whose point is centered at the remote spectrum monitor and that extends out in the direction of the interferer. The width indicates a measure of uncertainty in the location estimate.

VB: Public Width As [Integer](#)

C#: public [int](#) Width

C++: public [int](#) Width

Direction Field

If this is a directional antenna, perhaps a sector on a cell phone tower, for instance, this value indicates the direction angle from true north for the center line of the antenna sector. Due East is at 90 degrees.

VB: Public Direction As [Integer](#)

C#: public [int](#) Direction

C++: public [int](#) Direction

Height Field

The height location of the remote spectrum monitor.

VB: Public Height As [Integer](#)

C#: public [int](#) Height

C++: public [int](#) Height

Enumerations

PowerSource Enumeration

Designates the mode for Angle-of-Arrival and Power-on-Arrival location estimation.

Value:

Table 3-3. PowerSource Values and Definitions

Value	Definition
Power	User supplies Power values for each probe at the designated frequency
Trace	User supplies sweep traces for each probe. Vision extracts the power at the designated frequency.
Live	User supplies URLs for each probe. Vision captures sweep traces from each and extracts the power at the designated frequency.

3-4 Database Class

Methods

Database.AddChannel Method

Adds new channels to the list according to the parameters.

VB: Public Function AddChannel(ByVal *cInfo* As ChannelDescription) As Boolean

C#: public bool AddChannel(ChannelDescription *cInfo*)

C++: public bool AddChannel(ChannelDescription *cInfo*)

Parameters: *cInfo*: Channel description of the channel to add, contained in ChannelDescription structure.

Database.AddProbe Method

Enters a new probe into the database with the parameter *pInfo*.

VB: Public Function AddProbe(ByVal *pInfo* As ProbeDescription) As Boolean

C#: public bool AddProbe(ProbeDescription *pInfo*)

C++: public bool AddProbe(ProbeDescription *pInfo*)

Parameters: *pInfo*: Description of the probe to add, as contained in the ProbeDescription structure.

Database.CreateEmptyDatabase Method

Creates a new database set in parameters.

VB: Public Function CreateEmptyDatabase(ByVal *TargetPath* As String) As Boolean

C#: public bool CreateEmptyDatabase(string *TargetPath*)

C++: public bool CreateEmptyDatabase(string *TargetPath*)

Parameters: *TargetPath*: Full path in the local file system where the database is to be created. Do not include a file name. If a Vision database already exists at this location, it will be overwritten.

Database.DeleteChannel Method

Deletes the specified channel of the remote spectrum monitor set in parameters from the database.

VB: Public Function DeleteChannel(ByVal *ProbeName* As String,ByVal *ChannelNumber* As Integer) As Boolean

C#: public bool DeleteChannel(string *ProbeName*,int *ChannelNumber*)

C++: public bool DeleteChannel(string *ProbeName*,int *ChannelNumber*)

Parameters: *ProbeName*: Probe name containing the channel to delete.

ChannelNumber: The antenna port number on a multi-port target receiver.

Database.DeleteProbe Method

Deletes the specified remote spectrum monitor set in parameters from the database.

VB: Public Function DeleteProbe(ByVal *ProbeName* As String) As Boolean

C#: public bool DeleteProbe(string *ProbeName*)

C++: public bool DeleteProbe(string *ProbeName*)

Parameters: *ProbeName*: Name of the target receiver to delete from the database.

Database.GetChannel Method

Obtains the specified channel of the remote spectrum monitor set in parameters from the database.

VB: Public Function GetChannel(ByVal *ProbeName* As [String](#),ByVal *ChannelNumber* As [Integer](#)) As [ChannelDescription](#)

C#: public [ChannelDescription](#) GetChannel([string](#) *ProbeName*,[int](#) *ChannelNumber*)

C++: public [ChannelDescription](#) GetChannel([string](#) *ProbeName*,[int](#) *ChannelNumber*)

Parameters: *ProbeName*: Name of the target receiver for the desired channel.

ChannelNumber: Channel number to retrieve.

Database.GetDatabasePath Method

Obtains the database path.

VB: Public Function GetDatabasePath() As [String](#)

C#: public [string](#) GetDatabasePath()

C++: public [string](#) GetDatabasePath()

Database.GetProbe Method

Gets the specified remote spectrum monitor from the database set in parameters.

VB: Public Function GetProbe(ByVal *ProbeName* As [String](#)) As [ProbeDescription](#)

C#: public [ProbeDescription](#) GetProbe([string](#) *ProbeName*)

C++: public [ProbeDescription](#) GetProbe([string](#) *ProbeName*)

Parameters: *ProbeName*: Name of the probe to retrieve.

Database.GetTraces Method

Gets the traces of the specified channel and remote spectrum monitor set in parameters from the database.

VB: Public Function GetTraces(ByVal *ProbeName* As String,ByVal *Channel* As Integer) As ChannelData

C#: public ChannelData GetTraces(string *ProbeName*,int *Channel*)

C++: public ChannelData GetTraces(string *ProbeName*,int *Channel*)

Parameters: *ProbeName*: Name of a probe in the Vision database.

Channel: The channel to retrieve.

Database.ModifyChannel Method

Modifies the database content set in Parameters.

VB: Public Function ModifyChannel(ByVal *ProbeName* As String,ByVal *cInfo* As ChannelDescription) As Boolean

C#: public bool ModifyChannel(string *ProbeName*,ChannelDescription *cInfo*)cc

C++: public bool ModifyChannel(string *ProbeName*,ChannelDescription *cInfo*)

Parameters: *ProbeName*: Name of a probe in the Vision database.

clnfo: New channel description in a ChannelDescription structure.

Database.ModifyProbe Method

Modifies the probe definition set in Parameters.

VB: Public Function ModifyProbe(ByVal *pInfo* As ProbeDescription) As Boolean

C#: public bool ModifyProbe(ProbeDescription *pInfo*)

C++: public bool ModifyProbe(ProbeDescription *pInfo*)

Parameters: *plnfo*: A probeDescription structure containing the new probe parameters.

Database.ProbeList Method

Obtains the list of remote spectrum monitors from the database.

VB: Public Function ProbeList() As List(Of ProbeDescription)

C#: public List<ProbeDescription> ProbeList()

C++: public List<ProbeDescription> ProbeList()

Database.SetDatabasePath Method

Sets the location of the database set in parameters.

VB: Public Function SetDatabasePath(ByVal *Path* As String) As Boolean

C#: public bool SetDatabasePath(string *Path*)

C++: public bool SetDatabasePath(string *Path*)

Parameters: *Path*: Full path to the Vision database. This is the folder name only, do not include a file name.

Structures

ChannelData Structure

VB: Public Structure ChannelData
C#: public struct ChannelData
C++: public struct ChannelData

ProbeDescription Structure

VB: Public Structure ProbeDescription
C#: public struct ProbeDescription
C++: public struct ProbeDescription

Fields

CarrierFreq Field

Enter the carrier frequency.

VB: Public CarrierFreq As [Long](#)
C#: public [long](#) CarrierFreq
C++: public [long](#) CarrierFreq

Channel Field

Enter the channel number of the active remote spectrum monitor.

VB: Public Channel As [Integer](#)
C#: public [int](#) Channel
C++: public [int](#) Channel

Failed Field

Enters a failed status.

VB: Public Failed As [Integer](#)
C#: public [int](#) Failed
C++: public [int](#) Failed

Passed Field

Enters a passed status.

VB: Public Passed As [Integer](#)
C#: public [int](#) Passed
C++: public [int](#) Passed

ProbeName Field

Enter a name for the desired Remote Spectrum Analyzer.

```
VB: Public ProbeName As String
C#: public string ProbeName
C++: public string ProbeName
```

RefLevel Field

The Reference Level in dBm for the channel.

```
VB: Public RefLevel As Single()
C#: public float[] RefLevel
C++: public float[] RefLevel
```

Sector Field

The sector of an antenna in relation to North.

```
VB: Public Sector As Integer
C#: public int Sector
C++: public int Sector
```

StartFreq Field

The sweep Start Frequency in Hz.

```
VB: Public StartFreq As Double()
C#: public double[] StartFreq
C++: public double[] StartFreq
```

Status Field

Pass/Fail status when trace compared to mask.

```
VB: Public Status As Boolean()
C#: public bool[] Status
C++: public bool[] Status
```

StopFreq Field

The sweep Stop Frequency in Hz.

```
VB: Public StopFreq As Double()
C#: public double[] StopFreq
C++: public double[] StopFreq
```

Times Field

An array of time stamps. Each time stamp corresponds to the trace of the same index in the Traces array field of this structure.

```
VB: Public Times As String()
C#: public string[] Times
C++: public string[] Times
```

TraceCount Field

The trace sweep count useful for multiple-sweep measurements.

```
VB: Public TraceCount As Integer
C#: public int TraceCount
C++: public int TraceCount
```

TraceTable Field

The name of the trace table.

```
VB: Public TraceTable As String
C#: public string TraceTable
C++: public string TraceTable
```

Traces Field

Returns all traces in the database for the selected channel.

```
VB: Public Traces As String()
C#: public string[] Traces
C++: public string[] Traces
```

URL Field

Enter the URL of the selected remote spectrum monitor.

```
VB: Public URL As String
C#: public string URL
C++: public string URL
```

Structures**ChannelDescription Structure**

```
VB: Public Structure ChannelDescription
C#: public struct ChannelDescription
C++: public struct ChannelDescription
```

Fields**Active Field**

Boolean parameter that indicates if the channel is actively scanned by Vision Acquire.

```
VB: Public Active As Boolean
C#: public bool Active
C++: public bool Active
```

Averaging Field

Parameter that indicates if the channel is set to Averaging Mode.

VB: Public Averaging As [Integer](#)

C#: public [int](#) Averaging

C++: public [int](#) Averaging

Channel Field

Enter the channel number of the active remote spectrum monitor.

VB: Public Channel As [Integer](#)

C#: public [int](#) Channel

C++: public [int](#) Channel

ChannelDelay Field

A calibration setting for doing TDOA measurements.

VB: Public ChannelDelay As [Integer](#)

C#: public [int](#) ChannelDelay

C++: public [int](#) ChannelDelay

Direction Field

If this is a directional antenna, perhaps a sector on a cell phone tower, for instance, this value indicates the direction angle from true north for the center line of the antenna sector. Due East is at 90 degrees.

VB: Public Direction As [Integer](#)

C#: public [int](#) Direction

C++: public [int](#) Direction

FFT Field

Sets the FFT field for measurement configuration.

VB: Public FFT As [Boolean](#)

C#: public [bool](#) FFT

C++: public [bool](#) FFT

FailRate Field

The number of traces that failed.

VB: Public FailRate As [Single](#)

C#: public [float](#) FailRate

C++: public [float](#) FailRate

Frequency Field

A frequency for which the channel is purposed for.

VB: Public Frequency As [Double](#)

C#: public [double](#) Frequency

C++: public [double](#) Frequency

LastFail Field

Date and time of the last trace that failed.

```
VB: Public LastFail As Date
C#: public DateTime LastFail
C++: public DateTime LastFail
```

LastSweep Field

Date and time of last sweep.

```
VB: Public LastSweep As Date
C#: public DateTime LastSweep
C++: public DateTime LastSweep
```

MaskName Field

The name of the mask used to validate trace data in Vision Acquire. Masks are stored in the database, and the name used here is the name the mask has in the database.

```
VB: Public MaskName As String
C#: public string MaskName
C++: public string MaskName
```

Preamp Field

Sets the Preamp field for measurement configuration.

```
VB: Public Preamp As Boolean
C#: public bool Preamp
C++: public bool Preamp
```

ProbeName Field

Enter a name for the desired Remote Spectrum Analyzer.

```
VB: Public ProbeName As String
C#: public string ProbeName
C++: public string ProbeName
```

RBW Field

The Resolution Bandwidth setting for the channel.

```
VB: Public RBW As Double
C#: public double RBW
C++: public double RBW
```

RefLevel Field

The Reference Level in dBm for the channel.

```
VB: Public RefLevel As Single
C#: public float RefLevel
C++: public float RefLevel
```

RefLevelOffset Field

The Reference Level offset in dB.

VB: Public RefLevelOffset As [Single](#)

C#: public [float](#) RefLevelOffset

C++: public [float](#) RefLevelOffset

StartFreq Field

The sweep Start Frequency in Hz.

VB: Public StartFreq As [Double](#)

C#: public [double](#) StartFreq

C++: public [double](#) StartFreq

Status Field

Pass/Fail status when trace compared to mask.

VB: Public Status As [Boolean](#)

C#: public [bool](#) Status

C++: public [bool](#) Status

StopFreq Field

The sweep Stop Frequency in Hz.

VB: Public StopFreq As [Double](#)

C#: public [double](#) StopFreq

C++: public [double](#) StopFreq

TimeOut Field

A value that sets the number of seconds Vision Acquire waits for a probe to respond before moving on.

VB: Public TimeOut As [Integer](#)

C#: public [int](#) TimeOut

C++: public [int](#) TimeOut

TraceCount Field

The trace sweep count useful for multiple-sweep measurements.

VB: Public TraceCount As [Integer](#)

C#: public [int](#) TraceCount

C++: public [int](#) TraceCount

TraceMode Field

Sets the trace mode: Normal, Averaging, Max Hold, Min Hold.

VB: Public TraceMode As [Byte](#)

C#: public [byte](#) TraceMode

C++: public [byte](#) TraceMode

TracePoints Field

The number of data points in each sweep.

VB: Public TracePoints As Integer
C#: public int TracePoints
C++: public int TracePoints

VBW Field

The Video bandwidth setting for the channel.

VB: Public VBW As Double
C#: public double VBW
C++: public double VBW

ChannelCount Field

The number of channels int the channel list.

VB: Public ChannelCount As Integer
C#: public int ChannelCount
C++: public int ChannelCount

Contact Field

The email to those monitoring the remote spectrum monitor.

VB: Public Contact As String
C#: public string Contact
C++: public string Contact

Description Field

A description of the selected remote spectrum monitor.

VB: Public Description As String
C#: public string Description
C++: public string Description

ErrorCount Field

The number of traces that failed the mask for the selected channel.

VB: Public ErrorCount As Integer
C#: public int ErrorCount
C++: public int ErrorCount

GroupName Field

The name for a group of remote spectrum monitors.

VB: Public GroupName As String
C#: public string GroupName
C++: public string GroupName

Height Field

The height location of the remote spectrum monitor.

VB: Public Height As [Integer](#)

C#: public [int](#) Height

C++: public [int](#) Height

Hostname Field

Enter the hostname or URL address for the instrument taking measurements.

VB: Public Hostname As [String](#)

C#: public [string](#) Hostname

C++: public [string](#) Hostname

LastFail Field

Date and time of the last trace that failed.

VB: Public LastFail As [Date](#)

C#: public [DateTime](#) LastFail

C++: public [DateTime](#) LastFail

Latitude Field

The latitude coordinate of the remote spectrum monitor.

VB: Public Latitude As [Double](#)

C#: public [double](#) Latitude

C++: public [double](#) Latitude

Longitude Field

The longitude coordinate of the remote spectrum monitor.

VB: Public Longitude As [Double](#)

C#: public [double](#) Longitude

C++: public [double](#) Longitude

Name Field

A name given to a remote spectrum monitor.

VB: Public Name As [String](#)

C#: public [string](#) Name

C++: public [string](#) Name

Port Field

The HTTP port number for the desired remote spectrum monitor.

VB: Public Port As [Integer](#)

C#: public [int](#) Port

C++: public [int](#) Port

Status Field

Pass/Fail status when trace compared to mask.

VB: Public Status As [Byte](#)

C#: public [byte](#) Status

C++: public [byte](#) Status

TraceDatabase Field

A name given to a trace database.

VB: Public TraceDatabase As [String](#)

C#: public [string](#) TraceDatabase

C++: public [string](#) TraceDatabase

TraceTable Field

A name given to a trace table in the trace database.

VB: Public TraceTable As [String](#)

C#: public [string](#) TraceTable

C++: public [string](#) TraceTable

3-5 FileViewer Class

Properties

FileViewer.CPMChannelDescriptionList Property

Gets the channel description list for the currently loaded CPM file.

VB: Public ReadOnly Property CPMChannelDescriptionList() As List(Of String)

C#: public List<string> CPMChannelDescriptionList {get;}

C++: public List<string> CPMChannelDescriptionList {get;}

Property Value: Returns an array of strings.

Remarks

Use this property to return a list of valid Channel descriptions. Items from this list can then be feed back using the ChannelDescription Property to select a particular channel from a loaded CPM file.

FileViewer.ChannelDescription Property

Gets or sets a property to channel selection in CPM files..

VB: Public Property ChannelDescription() As String

C#: public string ChannelDescription {get; set;}

C++: public string ChannelDescription {get; set;}

Property Value: Type: String. Default is Null.

Remarks

When setting the ChannelDescription Property, a search is done in the channel list for the currently loaded CPM file. A matching channel is selected and the traces loaded into the trace view graph and the spectrogram.

To get a list of valid channel descriptions for the current file, use the CPMChannelDescriptionList method. This will return a list of valid descriptions. Selecting from this list will choose one of the current channels.

FileViewer.FileList Property

Gets the local file list as an array of strings.

VB: Public ReadOnly Property FileList() As List(Of String)

C#: public List<string> FileList {get;}

C++: public List<string> FileList {get;}

Property Value: Type: Array of strings. Default is Null

Remarks

Returns the list of target receiver (*.RSM) files listed in the File list under the LocalFile LocationType. Choosing from this list allows a valid entry for setting the Filename property to load the file and display the trace data.

FileViewer.Filename Property

Gets or sets the name of the current file loaded for display.

VB: Public Property Filename() As [String](#)

C#: public [string](#) Filename {get; set;}

C++: public [string](#) Filename {get; set;}

Property Value: Type: String. Null is no file is loaded; otherwise, the current filename.

Remarks

Setting the filename will load the file, choosing automatically the target receiver file mode, either RSM or CPM, based on the file extension.

When set, the file will be loaded into system memory. For target receiver RSM files, the trace will automatically display. For CPM files, the channel list will be populated. Selecting a channel from the list will display the trace(s) for the selected file and channel.

FileViewer.Folder Property

Gets or sets the default folder for target receiver RSM files on the local file system.

VB: Public Property Folder() As [String](#)

C#: public [string](#) Folder {get; set;}

C++: public [string](#) Folder {get; set;}

Property Value: Type: String. Defaults to the current user's document folder.

Remarks

The Folder property is also set in the User Control by pressing the Path button and selecting a folder in the browser dialog.

FileViewer.GetRemoteFolderList Property

Returns a list of of folders on the remote spectrum monitor

VB: Public ReadOnly Property GetRemoteFolderList() As [List\(Of String\)](#)

C#: public [List<string>](#) GetRemoteFolderList {get;}

C++: public [List<string>](#) GetRemoteFolderList {get;}

Remarks

Return list contains all subfolders of the current folder on the remote spectrum monitor. To navigate, get the list, find the folder in the list you want, then set that location using the RemoteFolder Property. This is an iterative process to go deep into the folder structure. You can set the full path at once, without the steps, if the full path is known before hand.

FileViewer.Hostname Property

Gets or sets the hostname or URL of a remote spectrum monitor.

VB: Public Property Hostname() As **String**

C#: public **string** Hostname {get; set;}

C++: public **string** Hostname {get; set;}

Property Value: Type: String. Default is null reference (**Nothing** in Visual Basic); otherwise, contains the hostname of the current remote spectrum monitor

Remarks

Use the Hostname property to set the host name or URL for a remote spectrum monitor. Setting this property allows FileViewer to retrieve remote file lists and remote trace files. If an invalid hostname is entered, or the host cannot be reached, connection efforts will time out.

FileViewer.LocationType Property

Gets or sets the location type. This is a value that indicates what type of files are to be loaded.

VB: Public Property LocationType() As **FileModes**

C#: public **FileModes** LocationType {get; set;}

C++: public **FileModes** LocationType {get; set;}

Property Value: Type: Vision.RSM.FileModes

Values: FileModes.CPMFile, FileModes.LocalFolder, and FileModes.RemoteRSM.

Remarks

This property has three values:

FileModes.CPMFile is used to load and display traces stored using the Vision High-Speed Port Scanner. These files typically contain many thousands of traces and may take several seconds to several minutes to load, depending on the size. Because CPM files contain traces collected from multiple probes and antenna ports, a channel list will be shown after the file has been loaded. Choosing a channel is necessary to display traces.

When choosing FileModes.LocalFolder, the side panel will display target receiver RSM files in the currently selected folder of the local file system. You can select one or more files in the file list to display the trace data.

Choosing FileModes.RemoteRSM sets the side panel controls for selecting a remote folder and file on a remote spectrum monitor. This can be used to view select files, or to download all of the files to the local file system.

FileViewer.RemoteFileList Property

Gets a complete list of all files in the current folder on the active remote spectrum monitor.

VB: Public ReadOnly Property RemoteFileList() As **List(Of String)**

C#: public **List<string>** RemoteFileList {get;}

C++: public **List<string>** RemoteFileList {get;}

Remarks

This list includes both measurement files (*.RSM) and setup files (*.stp)

FileViewer.RemoteFolder Property

Gets or sets the current folder on the active remote spectrum monitor.

VB: Public Property RemoteFolder() As **String**

C#: public **string** RemoteFolder {get; set;}

C++: public **string** RemoteFolder {get; set;}

Property Value: Type: String

Remarks

Remote folders on remote spectrum monitors always begin with a location specifier. For internal system memory, this is '/Internal'. For removable USB memory, the device name is assigned by the probe operation system when it is inserted. So you may have to query the probe for the USB memory device name using the GetRemoteFolderList method.

FileViewer.SelectAllRemote Property

Sets or clears the selection state of all files in the remote file list.

VB: Public Property SelectAllRemote() As **Boolean**

C#: public **bool** SelectAllRemote {get; set;}

C++: public **bool** SelectAllRemote {get; set;}

Property Value: **true** if all files in the list are selected; otherwise, **false**.

Remarks

Selected files are downloaded when the DownloadRemoteFiles method is called.

FileViewer.ShowSidePanel Property

Gets or sets a value indicating whether the side panel is shown in the User Control UI.

VB: Public Property ShowSidePanel() As **Boolean**

C#: public **bool** ShowSidePanel {get; set;}

C++: public **bool** ShowSidePanel {get; set;}

Property Value: Type: Boolean, **true** if the side panel is visible; otherwise, **false**.

Remarks

The side panel shows the location type and provides an interface for the user to select path and files for viewing traces. You will need to hide the side panel if you want to restrict the user from navigating the file system. This is useful in cases where the host program already provides file system browsing and you only want to use the File Viewer control for trace and spectrogram display.

FileViewer.ShowSpectrogram Property

Gets or sets a value indicating whether the Spectrogram graph is displayed.

VB: Public Property ShowSpectrogram() As [Boolean](#)

C#: public [bool](#) ShowSpectrogram {get; set;}

C++: public [bool](#) ShowSpectrogram {get; set;}

Property Value: Type: Boolean, **true** if the spectrogram graph is displayed; otherwise, **false**.

Remarks

If the spectrogram graph is hidden, the Trace display fills the entire user control area right of the side panel. If the side panel is also hidden, then the control is just a trace viewer.

FileViewer.SnapShot Property

Takes a snap shot of the user control screen image and returns as a bitmap.

VB: Public ReadOnly Property SnapShot() As [Bitmap](#)

C#: public [Bitmap](#) SnapShot {get;}

C++: public [Bitmap](#) SnapShot {get;}

Property Value: Type: Bitmap

Remarks

Image includes everything shown in the User Control, but not the containing application or form window. It is copied and returned as a standard Windows Bitmap.

Methods**FileViewer.Close Method**

Programmatically close the FileViewer control.

VB: Public Sub Close()

C#: public [System.Void](#) Close()

C++: public [System.Void](#) Close()

Remarks

Using Close stops any current action, saves current settings, and prepares the user control for program shutdown. If you do not call Close, current settings are not saved.

FileViewer.DownloadRemoteFiles Method

Initiates a download of files from the remote spectrum monitor to the location set in parameters.

VB: Public Sub DownloadRemoteFiles(ByVal *SaveFolder* As String)

C#: public System.Void DownloadRemoteFiles(string *SaveFolder*)

C++: public System.Void DownloadRemoteFiles(string *SaveFolder*)

Parameters: *SaveFolder*: Save location in the local file system for downloaded target receiver RSM files.

Remarks

Using Close stops any current action, saves current settings, and prepares the user control for program shutdown. If you do not call Close, current settings are not saved.

FileViewer.SetTraceData Method

Sets the trace data for viewing.

VB: Public Sub SetTraceData(ByVal *StartFrequency* As Long,ByVal *StopFrequency* As Long,ByVal *RefLevel* As Integer,ByVal *TraceArray* As String(),ByVal *TimeStamp* As Date(),ByVal *GPS* As PointF())

C#: public System.Void SetTraceData(long *StartFrequency*,long *StopFrequency*,int *RefLevel*,string[] *TraceArray*,DateTime[] *TimeStamp*,PointF[] *GPS*)

C++: public System.Void SetTraceData(long *StartFrequency*,long *StopFrequency*,int *RefLevel*,string[] *TraceArray*,DateTime[] *TimeStamp*,PointF[] *GPS*)

Parameters: *StartFrequency*: Start frequency in Hz for the traces.

StopFrequency: Stop frequency in Hz for the traces.

RefLevel: Reference Level in dBm.

TraceArray: An array of strings. Each element in the array is a list of powers describing the trace data. The power values can be delimited by comma, semicolon or tab characters. Comma is the expected delimiter, as this is what is returned by a remote spectrum monitor. However, the other delimiters are supported.

TraceArray(0) = "-89.54,-91.34,86.87,-79.19,..."

TraceArray(1) = "-87.14,-90.74,91.79,-84.39,..."

TraceArray(n) = "-90.43,-90.37,89.72,-87.39,..."

TimeStamp: An array of timestamps, each element corresponding to date and time of capture for the corresponding trace data element in the *TraceArray* array. If date and time are not available for the traces, this array may be empty. The time stamp is shown on the trace data plot if the information is provided.

GPS: An array of GPS coordinates, each element corresponding to location of capture for the corresponding trace data element in the *TraceArray* array. If GPS coordinates are not available for the traces, this array may be empty.

Remarks

The FileViewer object is typically used to display trace data in saved measurement files; either RSM files saved on a target receiver, or a collection of traces saved using the Vision High-Speed Port Scanner. *TraceArray* is used to circumvent the File requirement and display a list of traces from another source. This allows the FileViewer to be used as trace display object for traces acquired without using the standard Vision UI components.

Enumerations

FileModes Enumeration

Designates the file mode for the Vision FileViewer class.

Value:

Table 3-4. FileModes Values and Definitions

Value	Definition
RemoteRSM	File list comes for a remote spectrum analyzer. A URL must be provided.
LocalFolder	File list comes from the local disk space. Displays RSM files that have already been downloaded.
CPMFile	Trace list comes from a CPM file. CPM files are capture files from the Vision High-Speed Port Scanner. A CPM file may contain many traces acquired from multiple remote spectrum analyzers. The trace list lets the user select a particular remote probe/channel combination.

3-6 MaskEditor Class

Properties

MaskEditor.MaskName Property

Use this property to get/set the name of the current mask.

VB: Public Property MaskName() As String

C#: public string MaskName {get; set;}

C++: public string MaskName {get; set;}

Property Value: Type: String

Remarks

Returns the name of the current mask in the Mask Editor.

Methods

MaskEditor.GetDatabasePath Method

VB: Public Function GetDatabasePath() As [String](#)

C#: public [String](#) GetDatabasePath()

C++: public [String](#) GetDatabasePath()

Remarks

Returns the current database path in the local file system.

MaskEditor.GetMaskList Method

VB: Public Function GetMaskList() As List(Of[String](#))

C#: public List<[String](#)> GetMaskList()

C++: public List<[String](#)> GetMaskList()

Remarks

Returns a list of all masks in the active database. Each entry in the list contains two fields, delimited by a semi-colon. The first field indicates if the mask is segmented or continuous (Trace). The second field is the mask name.

MaskEditor.RemoveAMaskFromDB Method

VB: Public Sub RemoveAMaskFromDB(ByVal Maskname As [String](#))

C#: public [System.Void](#) RemoveAMaskFromDB([String](#) Maskname)

C++: public [System.Void](#) RemoveAMaskFromDB([String](#) Maskname)

Parameters: Maskname: The name of the mask to be removed.

Remarks

Removes the mask from the database. The Maskname parameter is not case sensitive. If not found, nothing is done.

MaskEditor.SetDatabasePath Method

VB: Public Function SetDatabasePath(ByVal Path As [String](#)) As [Boolean](#)

C#: public [bool](#) SetDatabasePath([String](#) Path)

C++: public [bool](#) SetDatabasePath([String](#) Path)

Parameters: Path: The full path in the local file system to the database you want to access with the MaskEditor API.

Remarks

Sets the path to the active database. This is the path only, and should not include a database name.

MaskEditor.addSegmentedMaskToDB Method

VB: Public Function addSegmentedMasktoDB(ByVal MaskName As [String](#),ByVal MaskDescription As [String](#)) As [Boolean](#)

C#: public [bool](#) addSegmentedMasktoDB([String](#) MaskName,[String](#) MaskDescription)

C++: public [bool](#) addSegmentedMasktoDB([String](#) MaskName,[String](#) MaskDescription)

Parameters: MaskName: The name of the mask to add. This name will identify the mask in the database and will be used to assign a particular mask to channels for Pass/Fail qualification.

MaskDescription: The full description of the mask. See Remarks below for the format.

Remarks

Maskname should be unique. Giving two masks in a database the same name will cause a conflict and you will not know which is being used.

MaskDescription is a string value the contains the mask segment parameters. The format is:

StartFreq_1, StopFreq_1, Level_1, UpperLower_1; StartFreq_2, StopFreq_2, Level_2, UpperLower_2; ...
;StartFreq_n, StopFreq_n, Level_n, UpperLower_n

Where UpperLower: 0 for Upper limit; 1 for lower limit

MaskEditor.addTraceMaskToDB Method

VB: Public Function addTraceMasktoDB(ByVal MaskName As [String](#),ByVal Freq1 As [Single](#),ByVal Freq2 As [Single](#),_ByVal TraceData As [String](#),ByVal UpperLimit As [Boolean](#))As [Boolean](#)

C#: public [bool](#) addTraceMasktoDB([String](#) MaskName,[float](#) Freq1,[float](#) Freq2, [String](#) TraceData, [bool](#) UpperLimit)

C++: public [bool](#) addTraceMasktoDB([String](#) MaskName,[float](#) Freq1,[float](#) Freq2, [String](#) TraceData, [bool](#) UpperLimit)

Parameters: MaskName: The name of the mask to add. This name will identify the mask in the database and will be used to assign a particular mask to channels for Pass/Fail qualification.

Freq1: The start frequency for the mask in Hz.

Freq2: The stop frequency of the mask in Hz.

TraceData: The mask trace data. A comma string containing power values in dBm used to qualify traces.

UpperLimit: Boolean value indicates if the mask is to be considered an upper limit (True) or a lower limit (False).

Remarks

Maskname should be unique. Giving two masks in a database the same name will cause a conflict and you will not know which is being used.

TraceData is a comma delimited string of power values in dBm. It is expected to have the same number of points as the trace DisplayPoints it will be used to test.

3-7 Monitor Class

Monitor.Database()

(Read Only) Returns path to the currently loaded database in Vision Monitor

VB: Public ReadOnly Property Database() As [String](#)

C#: public [string](#) Database {get;}

C++: public [string](#) Database {get;}

Monitor.TraceData()

(Read Only) Returns the current trace data displayed in the Vision Monitor Trace window. The return value is a string of amplitude values, delimited by commas.

VB: Public ReadOnly Property TraceData() As [String](#)

C#: public [string](#) TraceData {get;}

C++: public [string](#) TraceData {get;}

Monitor.TraceStartFrequency()

(Read Only) Returns the start frequency of the data displayed in the Vision Monitor Trace window. The return value will contain units (i.e. kHz, MHz, GHz)

VB: Public ReadOnly Property StartFrequency() As [String](#)

C#: public [string](#) StartFrequency {get; set;}

C++: public [string](#) StartFrequency {get; set;}

Monitor.TraceStopFrquency()

(Read Only) Returns the stop frequency of the data displayed in the Vision Monitor Trace window. The return value will contain units (i.e. kHz, MHz, GHz)

VB: ReadOnly Public Property StopFrequency() As [String](#)

C#: public [string](#) StopFrequency {get; set;}

C++: public [string](#) StopFrequency {get; set;}

Monitor.TraceRBW()

(Read Only) Returns the resolution bandwidth used for the data displayed in the Vision Monitor Trace window. The return value will contain units (i.e. kHz, MHz)

VB: ReadOnly Public Property TraceRBW() As [String](#)

C#: public [string](#) TraceRBW {get; set;}

C++: public [string](#) TraceRBW {get; set;}

Monitor.TraceMarker()

(Read Only) Returns the text displayed for the current marker. If no marker is set, returns an empty string. This is the line at the lower-left hand of the Trace Display shown when a marker is active. It will include frequency and power at the marker position, and also frequency and power at the delta marker position if available. If the Marker mode is Channel Power, then the marker holds the center frequency, channel power and peak power of the selected channel.

3-8 Occupancy Class

Properties

Occupancy.AntennaPort Property

Gets or sets the antenna port of the of the remote spectrum monitor.

```
VB: Public Property AntennaPort() As Integer
C#: public int AntennaPort {get; set;}
C++: public int AntennaPort {get; set;}
```

Occupancy.ArchiveFolder Property

Gets or sets the archive folder period.

```
VB: Public Property ArchiveFolder() As String
C#: public string ArchiveFolder {get; set;}
C++: public string ArchiveFolder {get; set;}
```

Occupancy.ChannelWidth Property

Gets or sets the channel width for the Occupancy report. Channel width is the frequency width of each sub-channel. The overall frequency span of the trace sweeps will be divided into bins of this width, and the power level in each bin will determine the occupancy for that channel.

```
VB: Public Property ChannelWidth() As Long
C#: public long ChannelWidth {get; set;}
C++: public long ChannelWidth {get; set;}
```

Occupancy.GroupByResult Property

Gets or sets the option for creating a group of sub-channels based of pass/fail status.

```
VB: Public Property GroupByResult() As Boolean
C#: public bool GroupByResult {get; set;}
C++: public bool GroupByResult {get; set;}
```

Occupancy.IncludeDetails Property

Gets Include Details status. If true, the occupancy report will be considerably longer, as more details will be included.

```
VB: Public Property IncludeDetails() As Boolean
C#: public bool IncludeDetails {get; set;}
C++: public bool IncludeDetails {get; set;}
```

Occupancy.Report Property

Creates the occupancy report and saves it under the filename specified in the input parameter. The extension of the file determines the file type -HTM or CSV.

VB: Public ReadOnly Property Report(ByVal *OutputFile* As String) As String

C#: [System.Runtime.CompilerServices.IndexerName("Report")]
public string this[string *OutputFile*] {get;}

C++: [System.Runtime.CompilerServices.IndexerName("Report")]
public string this[string *OutputFile*] {get;}

Parameters: *OutputFile*: The filename to save the report under. This may include a full path description. The file extension should be either .csv or .htm,. The extension determines the output format of the report.

Occupancy.StartFrequency Property

Gets or sets the start frequency at which to begin the occupancy report. This does not have to coincide with the start frequency of the trace data stored in the selected database(s).

VB: Public Property StartFrequency() As Long

C#: public long StartFrequency {get; set;}

C++: public long StartFrequency {get; set;}

Occupancy.StopFrequency Property

Gets or sets Stop Frequency at which to begin the occupancy report.

VB: Public Property StopFrequency() As Long

C#: public long StopFrequency {get; set;}

C++: public long StopFrequency {get; set;}

Occupancy.Threshold Property

Gets or sets the power threshold level.

VB: Public Property Threshold() As Single

C#: public float Threshold {get; set;}

C++: public float Threshold {get; set;}

Occupancy.ThresholdType Property

Gets or sets the Threshold Type as maximum, minimum or average power in the channel.

VB: Public Property ThresholdType() As Integer

C#: public int ThresholdType {get; set;}

C++: public int ThresholdType {get; set;}

Methods

Occupancy.LoadOwnerList Method

Sets the name for the Owner List in the Occupancy Report. The Owner List is the last column for each sub-channel. The Owner List is loaded from a file. The file is plain text, and should have two comma separated columns, the first column contains the frequency center for a sub-channel (in MHz). The second column contains the text description of that sub-channel.

The Owner List can be saved and loaded, but it is not stored in the Vision database, so you will have to reload it each time you want to create an occupancy report with channel owners listed.

It may be easiest to start the list using the Vision Occupancy Report generator. Simply click in the last column of the report after it is generated and enter the description. Once you save the report, you can edit it further in a text editor or spreadsheet, as long as the format is preserved when saved.

VB: Public Function LoadOwnerList(ByVal *Filename* As String) As String

C#: public string LoadOwnerList(string *Filename*)

C++: public string LoadOwnerList(string *Filename*)

Parameters: Filename: Filename of the owner list.

3-9 PowerOnArrival Class

Properties

PowerOnArrival.DegreeMinuteSecond Property

Gets or sets the format of GPS coordinate.

```
VB: Public Property DegreeMinuteSecond() As Boolean
C#: public bool DegreeMinuteSecond {get; set;}
C++: public bool DegreeMinuteSecond {get; set;}
```

Remarks

GPS coordinates are given in degrees. If PowerOfArrival.DegreeMinuteSecond is True, then the format is: dd° mm' ss.ss" If PowerOfArrival.DegreeMinuteSecond is False, then the format is in decimal, ie. dd.ddddd

PowerOnArrival.GPSCoordinates Property

Does an immediate query to the monitor for updated GPS location.

```
VB: Public Property GPSCoordinates(ByVal Index As Integer) As PointF
C#: [System.Runtime.CompilerServices.IndexerName("GPSCoordinates")]
    public PointF this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("GPSCoordinates")]
    public PointF this[int Index] {get; set;}
```

Parameters: Index: Power-on-Arrival uses three probes for its location estimation algorithm. This is the index of the probe you are addressing. Valid values are 0, 1, or 2.

PowerOnArrival.Hostname Property

Enter the hostname or URL address for the instrument taking the measurement.

```
VB: Public Property Hostname(ByVal Index As Integer) As String
C#: [System.Runtime.CompilerServices.IndexerName("Hostname")]
    public string this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("Hostname")]
    public string this[int Index] {get; set;}
```

Parameters: Index: Power-on-Arrival uses three probes for its location estimation algorithm. This is the index of the probe you are addressing. Valid values are 0, 1, or 2.

PowerOnArrival.Location Property

Gets the Power-On-Location estimate as a set of GPS coordinates.

VB: Public ReadOnly Property Location() As [POA_Result](#)
C#: public [POA_Result](#) Location {get;}
C++: public [POA_Result](#) Location {get;}

Remarks

This property may return an invalid or useless value if the measurement has not already been properly set up. Before reading the Location, you must set the Datasource and Interferer Frequency, at least.

PowerOnArrival.MeassuredPower Property

Gets or sets the measured power value set in Parameters.

VB: Public Property MeassuredPower(*ByVal Index* As [Integer](#)) As [Single](#)
C#: [System.Runtime.CompilerServices.IndexerName("MeassuredPower")]
public float this[int *Index*] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("MeassuredPower")]
public float this[int *Index*] {get; set;}

Parameters: Index: Power-on-Arrival uses three probes for its location estimation algorithm. This is the index of the probe you are addressing. Valid values are 0, 1, or 2.

PowerOnArrival.ReferenceLevel Property

Gets or sets the reference level amplitude value.

VB: Public Property ReferenceLevel() As [Single](#)
C#: public float ReferenceLevel {get; set;}
C++: public float ReferenceLevel {get; set;}

Remarks

This value is the display reference level, which means it has the reference level offset applied. It also means that a change to the reference level offset will change this setting (though the actual, unadjusted reference level will stay the same).

PowerOnArrival.SourceFrequency Property

Gets or sets the frequency of the source of interference.

VB: Public Property SourceFrequency() As [Double](#)
C#: public double SourceFrequency {get; set;}
C++: public double SourceFrequency {get; set;}

PowerOnArrival.SourceMode Property

Gets or sets the Source Mode: power at reference frequency, sweep traces from which Vision will extract the power at the source frequency, or live trace mode where Vision will extract the power at the source frequency.

VB: Public Property SourceMode() As [PowerSource](#)
C#: public [PowerSource](#) SourceMode {get; set;}
C++: public [PowerSource](#) SourceMode {get; set;}

PowerOnArrival.StartFrequency Property

Gets or sets the start frequency for the active remote spectrum monitor.

VB: Public Property StartFrequency() As Double

C#: public double StartFrequency {get; set;}

C++: public double StartFrequency {get; set;}

PowerOnArrival.StopFrequency Property

Gets or sets the Stop Frequency of the active remote spectrum monitor.

VB: Public Property StopFrequency() As Double

C#: public double StopFrequency {get; set;}

C++: public double StopFrequency {get; set;}

PowerOnArrival.Trace Property

Gets or sets the trace of the specified channel set in parameters from the database.

VB: Public Property Trace(ByVal *Index* As Integer) As String

C#: [System.Runtime.CompilerServices.IndexerName("Trace")]
public string this[int *Index*] {get; set;}

C++: [System.Runtime.CompilerServices.IndexerName("Trace")]
public string this[int *Index*] {get; set;}

Parameters: Index: Power-on-Arrival uses three probes for its location estimation algorithm. This is the index of the probe you are addressing. Valid values are 0, 1, or 2.

Methods

PowerOnArrival.VerifySource Method

Gets status of Verify Source to set the correct type of angle of arrival calculation depending on the source mode setting.

VB: Public Function VerifySource(ByVal *Index* As Integer) As Boolean

C#: public bool VerifySource(int *Index*)

C++: public bool VerifySource(int *Index*)

Parameters: *Index*: Power-on-Arrival uses three probes for its location estimation algorithm. This is the index of the probe you are addressing. Valid values are 0, 1, or 2.

Structures

POA_Result Structure

VB: Public Structure POA_Result

C#: public struct POA_Result

C++: public struct POA_Result

Fields

Latitude Field

The latitude coordinate of the remote spectrum monitor.

VB: Public Latitude As Single

C#: public float Latitude

C++: public float Latitude

Longitude Field

The longitude coordinate of the remote spectrum monitor.

VB: Public Longitude As Single

C#: public float Longitude

C++: public float Longitude

Text Field

Formatted GPS coordinates of the interferer.

VB: Public Text As String

C#: public string Text

C++: public string Text

3-10 RSM Class

The RSM Class provides a easy means for controlling a remote spectrum monitor. The properties and methods of this class map directly to the SCPI commands described in the instrument Programming Manual.

Since the properties and methods act directly on a remote spectrum monitor, you must first set the URL property before using other properties and methods.

Properties

RSM.AntennaPort Property

Gets or sets the active antenna port of multi-channel remote spectrum monitors such as MS27103A.

VB: Public Property AntennaPort() As Integer

C#: public int AntennaPort {get; set;}

C++: public int AntennaPort {get; set;}

RSM.Attenuation Property

Gets or sets the attenuation value of the remote spectrum monitor.

VB: Public Property Attenuation() As String

C#: public string Attenuation {get; set;}

C++: public string Attenuation {get; set;}

RSM.AudioMute Property

This property controls whether or not audio is output through the system speaker (default audio device) when doing AM.FM streaming.

VB: Public ReadOnly Property AudioMute() As Boolean

C#: public bool AudioMute {get;}

C++: public bool AudioMute {get;}

Remarks

Returns True if audio output is muted; False if audio output will be played.

RSM.AverageCount Property

Gets or sets the average count for use in calculating the average display value.

VB: Public Property AverageCount() As Integer

C#: public int AverageCount {get; set;}

C++: public int AverageCount {get; set;}

RSM.BWRatio Property

The ratio of VBW to RBW when VBW is in auto mode.

VB: Public Property BWRatio() As String

C#: public string BWRatio {get; set;}

C++: public string BWRatio {get; set;}

RSM.CenterFrequency Property

Gets or sets the center frequency of the active remote spectrum monitor.

VB: Public Property CenterFrequency() As [String](#)

C#: public [string](#) CenterFrequency {get; set;}

C++: public [string](#) CenterFrequency {get; set;}

Property Value: Type: String. Units are expected. If no units are provided, Hz is assumed.

Remarks

Setting the Center Frequency does not change the span, but does change both the start and stop frequencies. Allowed units are 'Hz', 'kHz', 'MHz' and 'GHz'. The Vision API verifies that the value is positive but does not verify the upper range, as this can vary across the support products. If the value is not valid, nothing is set. Reading back the value is the best way to verify success.

RSM.ChannelPower Property

Returns the Channel Power calculated for the most recent trace acquired from the active remote spectrum monitor.

VB: Public ReadOnly Property ChannelPower() As [String](#)

C#: public [string](#) ChannelPower {get;}

C++: public [string](#) ChannelPower {get;}

RSM.ConnectionOK Property

Gets a value indicating whether Vision API has an active connection to a remote spectrum monitor.

VB: Public ReadOnly Property ConnectionOK() As [Boolean](#)

C#: public [bool](#) ConnectionOK {get;}

C++: public [bool](#) ConnectionOK {get;}

Property Value: **true** if yes; otherwise, **false**.

Remarks

The return value indicates if the connection is valid and active. When true, this means the connection is useful, not that it is busy.

RSM.Detection Property

Gets or sets the detection method for calculating each display point.

VB: Public Property Detection() As [String](#)

C#: public [string](#) Detection {get; set;}

C++: public [string](#) Detection {get; set;}

Remarks

Each display point represents several measurements. The detection type determines how the display point is derived from its associated measurements.

RSM.DisplayPoints Property

Gets or sets the number of display points the remote spectrum monitor currently measures.

VB: Public Property DisplayPoints() As Integer

C#: public int DisplayPoints {get; set;}

C++: public int DisplayPoints {get; set;}

Remarks

Increasing the number of display points can improve the resolution of measurements but will also increase the sweep time.

RSM.FormatFrequency Property

Formats the frequency value as appropriate for display set in Parameters.

VB: Public ReadOnly Property FormatFrequency(ByVal *Freq* As Double, ByVal *Digits* As Integer) As String

C#: [System.Runtime.CompilerServices.IndexerName("FormatFrequency")]
public string this[double *Freq*, int *Digits*] {get;}

C++: [System.Runtime.CompilerServices.IndexerName("FormatFrequency")]
public string this[double *Freq*, int *Digits*] {get;}

Parameters: *Freq*: Frequency value in Hz.

Digits: Number of digits to display after the decimal.

RSM.FrequencySpan Property

Gets or sets the Frequency Span of the active remote spectrum monitor.

VB: Public Property FrequencySpan() As String

C#: public string FrequencySpan {get; set;}

C++: public string FrequencySpan {get; set;}

Property Value: Type: String. Units are expected. If no units are provided, then 'Hz' is assumed.

Remarks

Setting the Frequency Span changes the start and stop frequencies, but does not change the center frequency. The value must be positive.

RSM.FrequencyValue Property

Converts a formatted frequency value to a integer value in Hertz.

VB: Public ReadOnly Property FrequencyValue(ByVal *Freq* As String) As Long

C#: [System.Runtime.CompilerServices.IndexerName("FrequencyValue")]
public long this[string *Freq*] {get;}

C++: [System.Runtime.CompilerServices.IndexerName("FrequencyValue")]
public long this[string *Freq*] {get;}

Parameters: *Freq*: A string value including units. Units are not case-sensitive. Expected units are Hz, kHz, MHz, and GHz.

RSM.GPS Property

Obtains the latitude and longitude coordinates of the remote spectrum monitors.

VB: Public ReadOnly Property GPS() As [String](#)

C#: public [string](#) GPS {get;}

C++: public [string](#) GPS {get;}

RSM.GPSDisplay Property

Gets or sets a parameter that determines how GPS coordinates are displayed.

VB: Public Property GPSDisplay() As [GPSDisplayMode](#)

C#: public [GPSDisplayMode](#) GPSDisplay {get; set;}

C++: public [GPSDisplayMode](#) GPSDisplay {get; set;}

Property Value: Either [GPSDisplayMode.GPS_Decimal](#) or [GPSDisplayMode.GPS_DMS](#)

Remarks

Sets whether GPS coordinates are displayed as decimal numbers, or with degree-minute-second format. This is not a setting on the remote spectrum monitor. It affects the Vision API behavior only.

RSM.IQBandwidth Property

Gets or sets the I/Q capture bandwidth.

VB: Public Property IQBandwidth() As [String](#)

C#: public [string](#) IQBandwidth {get; set;}

C++: public [string](#) IQBandwidth {get; set;}

Property Value: Type: String. The value is expected to carry units (kHz or MHz). If no units are included, Hz is assumed.

The allowed values are limited to a redefined set. These are: 13.3 kHz, 26.7 kHz, 66.7 kHz, 133 kHz, 267 kHz, 667 kHz, 1.33 MHz, 2.67 MHz, 6.67 MHz, 13.3 MHz and 20 MHz. If you try to set a different value, the next higher value is chosen. So if you try to set it to 200 kHz, you will get 267 kHz.

Remarks

Setting the bandwidth also sets the sample rate. Higher bandwidths use higher sample rates.

You can refer to the UG for the remote spectrum monitor for the exact correlation to bandwidth and sample rates. Generally, the sample rate is about equal to the bandwidth. For example, a bandwidth of 133 kHz has a sample rate of 150,000 I/Q samples/second. A bandwidth of 20 MHz has a sample rate of about 25,000,000 sample per second. So increasing the sample rate greatly increases the number of bytes that have to be transferred and stored when doing an I/Q capture. Consequently, the capture length can be constrained by the bandwidth. At lower capture bandwidths you can do 10 second captures. At higher bandwidths this is not possible. At 20 MHz, 2 seconds is the longest allowed capture length.

RSM.IQBitDepth Property

Gets or sets the I/Q capture bit resolution.

VB: Public Property IQBitDepth() As [BitDepth](#)

C#: public [BitDepth](#) IQBitDepth {get; set;}

C++: public [BitDepth](#) IQBitDepth {get; set;}

Property Value: Type: Vision.RSM.BitDepth

Allowed values are: Vision.RSM.BitDepth.Eight, Vision.RSM.BitDepth.Ten, Vision.RSM.BitDepth.Sixteen, Vision.RSM.BitDepth.TwentyFour

Remarks

Setting the bit depth affects several aspects of the I/Q capture process.

First, the higher the bit resolution, the larger the capture size in bytes. When compared to an 8 bit capture, a 24 bit capture takes 3 times as long to download from the probe, and also occupies 3 times as much disk space. The remote probes have a fixed amount of memory for storing I/Q capture data, so higher resolution may affect available capture lengths.

Second, a higher bit resolution greatly increases the integrity of the captured signal. An 8 bit capture usually is insufficient for a useful capture. If the reference level is not set near to the peak level of the signal being captured, then the effective resolution is lower, as several of the higher bits will never be used. It isn't unusual in an 8 bit capture to only use 4 bits for the actual data because of where the reference level is set. When this is plotted, the I/Q pairs will form a series of columns and rows on the plot showing the very low resolution of the actual data capture. In most cases, 16 bit is the best trade-off between data integrity and size/speed considerations.

When doing TDOA, one of the bits in each sample is used to carry timing information. For 10 and 24 bit depths, there are extra bits available and the resolution is not affected. For 8 and 16 bit depths, turning on the time stamping effectively lowers the resolution by one bit.

RSM.IQCaptureLength Property

Gets or sets the time length of an I/Q capture. Input is in milliseconds

VB: Public Property IQCaptureLength() As [Integer](#)

C#: public [int](#) IQCaptureLength {get; set;}

C++: public [int](#) IQCaptureLength {get; set;}

Property Value: Type: Integer. The number of milliseconds to capture.

Remarks

Capture length is directly proportional to capture size. The allowed length is constrained to some degree by both the bit solution and by the capture bandwidth.

RSM.IQ_I Property

Gets the I values following an I/Q capture.

VB: Public ReadOnly Property IQ_I() As Integer()

C#: public int[] IQ_I {get;}

C++: public int[] IQ_I {get;}

Property Value: Type: Integer

Remarks

Following a successful I/Q capture, use IQ_I to return a reference to the integer array hold the I values. Use IQ_I.Count to get the number of samples. Use IQ_I.Clone to make a deep copy.

RSM.IQ_Plot Property

Returns a bitmap of the IQ values plotted with I on the horizontal axis and Q on the vertical. This must follow a call to IQCapture.

VB: Public ReadOnly Property IQ_Plot(ByVal Width As Integer,ByVal Height As Integer) As Bitmap

C#: [System.Runtime.CompilerServices.IndexerName("IQ_Plot")]
public Bitmap this[int Width,int Height] {get;}

C++: [System.Runtime.CompilerServices.IndexerName("IQ_Plot")]
public Bitmap this[int Width,int Height] {get;}

Parameters: Width: The width of the resultant bitmap.

Height: The height of the resultant bitmap.

Property Value: Type: Bitmap

Remarks

Typically, you will define a target, such as a picture box or panel that you want to assign the bitmap to, using either the Image property or the BackgroundImage property of the target control. Then you can use `PictureBox.Image = Probe.IQPlot()`.

Since the IQPlot method returns a reference to a standard Windows Bitmap, you can use it anywhere in code where you might use a bitmap reference. To save directly to file, you can use the Save property of the Bitmap object. For example, `Probe.IQPlot().Save(Filename, Filetype)`.

Calling IQPlot must follow a call to IQCapture.

RSM.IQ_Q Property

Gets the Q values following an I/Q capture.

```
VB: Public ReadOnly Property IQ_Q() As Integer()  
C#: public int[] IQ_Q {get;}  
C++: public int[] IQ_Q {get;}
```

Remarks

Following a successful I/Q capture, use IQ_Q to return a reference to the integer array hold the Q values. Use IQ_Q.Count to get the numbe of samples.

Use IQ_Q.Clone to make a deep copy.

RSM.MeasurementStatus Property

Use this property to get the status of FM Deviation or AM Modulation measurements.

```
VB: Public ReadOnly Property MeasurementStatus() As Boolean  
C#: public bool MeasurementStatus {get;}  
C++: public bool MeasurementStatus {get;}
```

Remarks

Returns True if either FM Deviation or AM Modulation Index measurements are in progress; False otherwise.

RSM.OBW Property

Returns the Occupied Bandwidth for the most recent trace acquired from the active remote spectrum monitor.

```
VB: Public ReadOnly Property OBW() As String  
C#: public string OBW {get;}  
C++: public string OBW {get;}
```

RSM.OBWPercent Property

Gets or sets the percentage of carrier power used to measure the occupied bandwidth.

```
VB: Public Property OBWPercent() As Integer  
C#: public int OBWPercent {get; set;}  
C++: public int OBWPercent {get; set;}
```

RSM.Options Property

Options are returned as a string of 3 digit numbers, separated by a forward slash character (“/”).

```
VB: Public ReadOnly Property Options() As String  
C#: public string Options {get;}  
C++: public string Options {get;}
```

Remarks

Returns installed options on the indicated RSM.

RSM.Preamp Property

Gets and sets the Preamp On/Off.

VB: Public Property Preamp() As Integer

C#: public int Preamp {get; set;}

C++: public int Preamp {get; set;}

RSM.ProductID Property

Gets the assembly name and version number for the Vision API dll assembly.

VB: Public ReadOnly Property ProductID() As String

C#: public string ProductID {get;}

C++: public string ProductID {get;}

Property Value: Type: String.

Remarks

Used this property to verify the version number of the installed Vision files.

RSM.RBW Property

Gets or sets the Resolution Bandwidth for the active remote spectrum monitor.

VB: Public Property RBW() As String

C#: public string RBW {get; set;}

C++: public string RBW {get; set;}

Property Value: Type: String

Remarks

RBW values are restricted to a predefined set. Allowed values are: 3 Hz, 10 Hz, 30 Hz, 100 Hz, 300 Hz, 1 kHz, 3 kHz, 10 kHz, 30 kHz, 100 kHz, 300 kHz, 1 MHz, 3 MHz.

RSM.RBWAuto Property

Gets or sets the Auto RBW state of the remote spectrum monitor

VB: Public Property RBWAuto() As String

C#: public string RBWAuto {get; set;}

C++: public string RBWAuto {get; set;}

Property Value: Type: String

Remarks

The return value is either 'On' or 'Off'. Use these same string values for setting the state.

RSM.RBWRatio Property

Gets or sets the RBW ratio.

VB: Public Property RBWRatio() As String

C#: public string RBWRatio {get; set;}

C++: public string RBWRatio {get; set;}

RSM.ReferenceLevel Property

Gets or sets the reference level amplitude value.

VB: Public Property ReferenceLevel() As [String](#)

C#: public [string](#) ReferenceLevel {get; set;}

C++: public [string](#) ReferenceLevel {get; set;}

Remarks

This value is the display reference level, which means it has the reference level offset applied. It also means that a change to the reference level offset will change this setting (though the actual, unadjusted reference level will stay the same).

RSM.ReferenceLevelOffset Property

Gets or sets the reference level offset value for the y-axis.

VB: Public Property ReferenceLevelOffset() As [String](#)

C#: public [string](#) ReferenceLevelOffset {get; set;}

C++: public [string](#) ReferenceLevelOffset {get; set;}

Remarks

This offset is used for display purposes only, and does not affect the actual reference level used for auto attenuation coupling, or any other settings that depend on reference level.

RSM.StartFrequency Property

Gets or sets the start frequency for the active remote spectrum monitor.

VB: Public Property StartFrequency() As [String](#)

C#: public [string](#) StartFrequency {get; set;}

C++: public [string](#) StartFrequency {get; set;}

Property Value: Type: String. Units are expected. If no units are provided, 'Hz' is assumed.

Remarks

Setting the Start Frequency also changes the span and center frequency. It does not change the stop frequency. Frequency inputs are ranged checked to be positive, but no upper range is verified as the frequency range varies across supported products. **Remark**se best way to verify the setting is to read back the value.

RSM.StopFrequency Property

Gets or sets the Stop Frequency of the active remote spectrum monitor.

VB: Public Property StopFrequency() As [String](#)

C#: public [string](#) StopFrequency {get; set;}

C++: public [string](#) StopFrequency {get; set;}

Property Value: Type: String. Units are expected. If no units are provided, 'Hz' is assumed.

Remarks

Setting the Stop Frequency also changes the span and center frequency. It does not change the start frequency. Frequency inputs are ranged checked to be positive, but no upper range is verified as the frequency range varies across supported products. The best way to verify the setting is to read back the value.

RSM.Streaming Property (Read Only)

AM/FM streaming is a continuous process that once started, continues until it is either stopped by the RSM.StopStreaming() method or by a change in sweep parameters on the RSM. To determine if the RSM is still streaming audio, check this property.

VB: Public Property Streaming() As [Boolean](#)

C#: public [bool](#) Streaming {get; set;}

C++: public [bool](#) Streaming {get; set;}

Remarks

True is AM or FM audio is currently streaming; False otherwise.

RSM.SweepMode Property

Gets or sets the type of sweep mode for active remote spectrum monitor. Modes are Single Trace, All Traces, Min Hold Trace, Max Hold Trace, and Average Trace.

VB: Public Property SweepMode() As [String](#)

C#: public [string](#) SweepMode {get; set;}

C++: public [string](#) SweepMode {get; set;}

RSM.Sweeping Property

Checks the status of Sweeping.

VB: Public Property Sweeping() As [Boolean](#)

C#: public [bool](#) Sweeping {get; set;}

C++: public [bool](#) Sweeping {get; set;}

RSM.Trace Property

Gets the trace of the remote spectrum monitor.

VB: Public ReadOnly Property Trace() As [String](#)

C#: public [string](#) Trace {get;}

C++: public [string](#) Trace {get;}

RSM.TraceMode Property

Gets or sets the type of sweep mode for active remote spectrum monitor. Modes are Normal Trace, Rolling Min, Rolling Averaging, Rolling Max, and Trace Count.

VB: Public Property TraceMode() As [String](#)

C#: public [string](#) TraceMode {get; set;}

C++: public [string](#) TraceMode {get; set;}

RSM.URL Property

Gets or sets the URL of the desired remote spectrum monitor.

VB: Public Property URL() As [String](#)

C#: public [string](#) URL {get; set;}

C++: public [string](#) URL {get; set;}

Property Value: Type: String. Default value is null reference (**Nothing** in Visual Basic)

Remarks

This property must be set before other methods and property can be used in any meaningful way.

The URL can be an IP address (xxx.xxx.xxx.xxx) or a host name registered with a domain name server. It is the same name for the probe that would be used if accessing the probe via the web browser interface.

RSM.Units Property

Gets or sets the units of the desired measurement type.

VB: Public Property Units() As [String](#)

C#: public [string](#) Units {get; set;}

C++: public [string](#) Units {get; set;}

RSM.VBW Property

Gets or sets video bandwidth of the active remote spectrum monitor.

VB: Public Property VBW() As [String](#)

C#: public [string](#) VBW {get; set;}

C++: public [string](#) VBW {get; set;}

RSM.VBWAuto Property

Gets or sets the Auto VBW state of the remote spectrum monitor.

VB: Public Property VBWAuto() As [String](#)

C#: public [string](#) VBWAuto {get; set;}

C++: public [string](#) VBWAuto {get; set;}

Property Value: Type: String.

Remarks

The return value is either 'On' or 'Off'. Use these same string values for setting the state.

RSM.VBWType Property

Gets or sets VBW Tpye for the active remote spectrum monitor.

VB: Public Property VBWType() As [String](#)

C#: public [string](#) VBWType {get; set;}

C++: public [string](#) VBWType {get; set;}

Property Value: Type: String. Either 'Linear' or 'Logarithmic'

RSM.Version Property

Obtains the application version.

VB: Public ReadOnly Property Version() As [String](#)

C#: public [string](#) Version {get;}

C++: public [string](#) Version {get;}

RSM.YScale Property

Gets or sets the number of dBm/division when plotting sweep trace data.

VB: Public Property YScale() As [Integer](#)

C#: public [int](#) YScale {get; set;}

C++: public [int](#) YScale {get; set;}

Remarks

This value is not sent to the remote spectrum monitor and does not affect its display. The Vision API generates its own trace graphs from trace data downloaded from the remote probe. This value is internal only. Valid values range from 1 to 20 dBm/Division.

Methods

RSM.AMModulation Method

Sets the frequency of the RSM and an appropriate bandwidth for an AM signal. Measures the AM Modulation for the indicated time, then returns a numeric value indicating the Modulation Index as a percentage.

(ByValue CarrierFrequency as Long, ByVal RunTimeSeconds as Integer)

Input Fields

VB: Public AmModulation(ByVal *CarrierFrequency* As String,ByVal *Integer*,ByVal *RunTimeSeconds* Integer)

C#: public AmModulation(string *CarrierFrequency*,int *RunTimeSeconds*,int

C++: public AmModulation(string *CarrierFrequency*,int *RunTimeSeconds*,int

Remarks

Input Fields

CarrierFrequency (String): The carrier frequency of the signal to measure. Units are allowed. If units are not included, Hz is assumed.

RunTimeSeconds (Integer): The number of seconds to perform the measurement.

Return Type: SingleReturns the AM modulation index at the set frequency, measured over the set time, as a percentage.

RSM.CaptureIQ Method

Initiates IQ capture on the current probe. When complete, the I/Q sample pairs are available using the IQ_I and IQ_Q properties of the class.

VB: Public Function CaptureIQ() As Boolean

C#: public bool CaptureIQ()

C++: public bool CaptureIQ()

Returns: Type: Boolean. Returns **true** if the IQ capture is successful; otherwise, returns **false**.

Remarks

Before using the CaptureIQ method, you must first set the following properties: center frequency, reference level, IQBitDepth, IQBandwidth, & IQCaptureLength. Failing to do so will produce undefined results, meaning previous or default values may be used. If you are making repeat measurements on the same probe and with the same capture parameters, you do not need to reset them between each call to IQCapture. However, if another remote user accesses the probe, the parameters may change.

On a multi-channel probe, be sure to set the correct antenna channel.

This function may take several seconds to return, depending on the capture length, bit depth and bandwidth, and also on the Internet connection bandwidth. I/Q captures can produce very large result arrays, so the download time will be long on a slow network connection. For instance, a 16 bit capture at 20 MHz for 1 second produces 100 MBytes of data.

RSM.CheckOption Method

VB: Public Function CheckOption(ByVal As [String](#)) Return Type As [Boolean](#)

C#: public [string](#) CheckOption as([bool](#))

C++: public [string](#) CheckOption as([bool](#))

Return Value

True if the option is installed; False if it is not installed.

RSM.CreatePlayBackWaveFile

When streaming AM or FM audio, small (~ 3 second) sound chunks are received and stored as .wav files in a temporary folder on your PC. This method combines these files into a single output file and moves it to the location of your choice. The smaller files are deleted from the temporary folder.

There is a program named Lame.exe that is included in the application folder. If this program is present, then it is used to convert the .wav output into mp3 format. The .wav files are uncompressed audio, and take up a lot of space. The mp3 files are much more compact. If you intended to capture a lot of audio, you will want to use the compressed format.

VB: CreatePlayBackWaveFile([string](#) *Filename*)

C#: CreatePlayBackWaveFile([string](#) *Filename*)

C++: CreatePlayBackWaveFile([string](#) *Filename*)

RSM.DeleteFile Method

Deletes the file set in parameters.

VB: Public Function DeleteFile(ByVal *RemoteFilename* As [String](#)) As [Boolean](#)

C#: public [bool](#) DeleteFile([string](#) *RemoteFilename*)

C++: public [bool](#) DeleteFile([string](#) *RemoteFilename*)

Parameters: RemoteFilename: Fully qualified path name. Network paths must be mapped to a drive letter.

RSM.DisplayStoredTrace Method

Returns a bitmap image plotting the trace data in the requested file on the remote spectrum monitor.

VB: Public Function DisplayStoredTrace(*ByVal RemoteFilename* As [String](#),*ByVal Width* As [Integer](#),*ByVal Height* As [Integer](#)) As [Bitmap](#)

C#: public [Bitmap](#) DisplayStoredTrace([string RemoteFilename](#),[int Width](#),[int Height](#))

C++: public [Bitmap](#) DisplayStoredTrace([string RemoteFilename](#),[int Width](#),[int Height](#))

Parameters: RemoteFilename: Filename of an RSM file on a remote spectrum monitor.

Width: Desired width, in pixels, of the resultant bitmap.

Height: Desired height, in pixels, of the resultant bitmap.

RSM.DownloadFile Method

Downloads a file into the database set by parameters.

VB: Public Function DownloadFile(*ByVal RemoteFilename* As [String](#),*ByVal LocalPath* As [String](#)) As [Boolean](#)

C#: public [bool](#) DownloadFile([string RemoteFilename](#),[string LocalPath](#))

C++: public [bool](#) DownloadFile([string RemoteFilename](#),[string LocalPath](#))

Parameters: RemoteFilename: Filename of an RSM file on a remote spectrum monitor.

LocalPath: It is the local directory path where you want the remote file to be stored.

RSM.GetFiles Method

Returns the list of files in the Remote Folder.

VB: Public Function GetFiles(*ByVal RemoteFolder* As [String](#)) As [String](#)()

C#: public [string](#)[] GetFiles([string RemoteFolder](#))

C++: public [string](#)[] GetFiles([string RemoteFolder](#))

Parameters: RemoteFolder: Folder on the RSM. This folder is expected to exist and contain 1 or more measurement files (*.RSM).

RSM.GetFolders Method

Returns the list of folders in Remote Folder.

VB: Public Function GetFolders(*ByVal RemoteFolder* As [String](#)) As [String](#)()

C#: public [string](#)[] GetFolders([string RemoteFolder](#))

C++: public [string](#)[] GetFolders([string RemoteFolder](#))

Parameters: RemoteFolder: Folder on the RSM. This folder is expected to exist. The return will be a list of sub-folders to the folder provided.

RSM.NewFolder Method

If true, enter a new folder as set by Parameters.

VB: Public Function NewFolder(*ByVal RemoteFolder* As [String](#)) As [Boolean](#)

C#: public [bool](#) NewFolder([string RemoteFolder](#))

C++: public [bool](#) NewFolder([string RemoteFolder](#))

Parameters: RemoteFolder: Folder on the RSM. This folder is expected to exist and a new sub-folder will be created within.

RSM.PlayIQ_AM Method

This attempts to do AM demodulation of the current I/Q capture data and play it on the system speaker.

```
VB: Public Sub PlayIQ_AM()  
C#: public System.Void PlayIQ_AM()  
C++: public System.Void PlayIQ_AM()
```

Remarks

This can produce decent output if you have captured an AM signal, with appropriate depth and bandwidth. For best results, set the reference level within 10 dB of the signal peak height.

RSM.PlayIQ_FM Method

This attempts to do an FM demodulation of the current I/Q capture data and play it on the system speaker.

```
VB: Public Sub PlayIQ_FM()  
C#: public System.Void PlayIQ_FM()  
C++: public System.Void PlayIQ_FM()
```

Remarks

This can produce decent output if you have captured an FM signal, with appropriate depth and bandwidth. Wide-band FM typically works best with 267 khz capture bandwidth at 16 bit resolution. You can capture upto 5 seconds of audio with these parameters. For narrow-band FM, you can capture longer. For best results, set the reference level within 10 dB of the signal peak height.

RSM.Preset Method

Presets all traces.

```
VB: Public Sub Preset()  
C#: public System.Void Preset()  
C++: public System.Void Preset()
```

RSM.Reboot Method

Reboots the selected remote spectrum monitor.

```
VB: Public Sub Reboot()  
C#: public System.Void Reboot()  
C++: public System.Void Reboot()
```

RSM.RecallMeasurement Method

Lists all of the measurements in the file set by Parameters.

```
VB: Public Function RecallMeasurement(ByVal RemoteFilename As String) As Boolean  
C#: public bool RecallMeasurement(string RemoteFilename)  
C++: public bool RecallMeasurement(string RemoteFilename)
```

Parameters: RemoteFilename: The name of a measurement file on the remote spectrum analyzer.

RSM.ResetCmdPort Method

Resets the SCPI command port on a remote spectrum monitor

VB: Public Sub ResetCmdPort()

C#: public System.Void ResetCmdPort()

C++: public System.Void ResetCmdPort()

Remarks

Under unusual circumstances, the Ethernet port that receives and processes SCPI commands on the remote probe can get stuck. When this occurs, this API command may remove the obstruction.

RSM.SaveIQ Method

Save I/Q capture data to disk file.

VB: Public Sub SaveIQ(ByVal *Filename* As String)

C#: public System.Void SaveIQ(string *Filename*)

C++: public System.Void SaveIQ(string *Filename*)

Parameters: *Filename*: Filename where the data is to be saved. This includes full path information. Network paths must be mapped to a drive letter.

Remarks

Existing files will be overwritten. If you want overwrite protection, provide that in the parent application. If you write to a protected folder, or try to overwrite an open file, an error will be returned.

Output format is ASCII. Two columns of integer values (I, Q) tab delimited, with each line ending in carriage return/line feed.

RSM.SaveIQ_AM Method

Save AM demodulated audio to a Windows .wav file for future playback.

VB: Public Sub SaveIQ_AM(ByVal *Filename* As String)

C#: public System.Void SaveIQ_AM(string *Filename*)

C++: public System.Void SaveIQ_AM(string *Filename*)

Parameters: *Filename*: Fully qualified path name. Network paths must be mapped to a drive letter.

Remarks

An existing file will be overwritten. Overwrite protection must be implemented in the parent application.

RSM.SaveIQ_FM Method

Save FM demodulated audio to a Windows .wav file.

VB: Public Sub SaveIQ_FM(ByVal *Filename* As String)

C#: public System.Void SaveIQ_FM(string *Filename*)

C++: public System.Void SaveIQ_FM(string *Filename*)

Parameters: *Filename*: Fully qualified path name. Network paths must be mapped to a drive letter.

Remarks

An existing file will be overwritten. Overwrite protection must be implemented in the parent application.

RSM.SaveMeasurement Method

Initiates a save measurement set by parameters.

VB: Public Function SaveMeasurement(ByVal *RemoteFilename* As String) As Boolean

C#: public bool SaveMeasurement(string *RemoteFilename*)

C++: public bool SaveMeasurement(string *RemoteFilename*)

Parameters: RemoteFilename: The filename under which to save the measurement file. A folder can be specified as part of the filename. If a folder is given, that folder is expected to exist.

RSM.SaveSetup Method

Initiates a save configuration setup set by parameters.

VB: Public Function SaveSetup(ByVal *RemoteFilename* As String) As Boolean

C#: public bool SaveSetup(string *RemoteFilename*)

C++: public bool SaveSetup(string *RemoteFilename*)

Parameters: RemoteFilename: The filename under which to save the current instrument setup. A folder can be specified as part of the filename. If a folder is given, that folder is expected to exist.

RSM.SendCommand Method

This method provides a means to send a SCPI command directly to the remote spectrum monitor. Although the properties and methods of this class cover most SCPI commands, there may be occasions where a user wants direct control to send exactly a sequence of Commands, or to send a command that is not mapped in the RSM Class. This method provides a simple interface to send any command detailed in the RSM Programming Manual directly to the RSM.

RSM.SweepGraph Method

Returns an image as a Windows Bitmap with the height and width as specified in the Parameters. This image is the Trace sweep graphed in standard X-Y coordinates.

VB: Public Function SweepGraph(ByVal *Width* As Integer,ByVal *Height* As Integer) As Bitmap

C#: public Bitmap SweepGraph(int *Width*,int *Height*)

C++: public Bitmap SweepGraph(int *Width*,int *Height*)

Parameters: Width: Width of the return bitmap in pixels.

Height: Height of the return bitmap in pixels

RSM.SweepGraph64 Method

Returns an image as a Base64 encoded string of a Windows Bitmap, with the height and width as specified in the Parameters. This image is the Trace sweep graphed in standard X-Y coordinates.

VB: Public Function SweepGraph64(ByVal *Width* As Integer,ByVal *Height* As Integer) As String

C#: public string SweepGraph64(int *Width*,int *Height*)

C++: public string SweepGraph64(int *Width*,int *Height*)

Parameters: Width: Width of the return bitmap in pixels.

Height: Height of the return bitmap in pixels

RSM.TriggerSweep Method

In single sweep mode, triggers a sweep.

VB: Public Sub TriggerSweep()
C#: public [System.Void](#) TriggerSweep()
C++: public [System.Void](#) TriggerSweep()

Events

OnFrequencyChange Event

Returns a notification that the sweep parameters have changed in the remote spectrum analyzer.

VB: Public Event OnFrequencyChange(ByVal *FrequencyArgs* As [FrequencyParameters](#))
C#: public event OnFrequencyChangeEventHandler OnFrequencyChange
 delegate void OnFrequencyChangeEventHandler([FrequencyParameters](#) *FrequencyArgs*)
C++: public event OnFrequencyChangeEventHandler OnFrequencyChange
 delegate void OnFrequencyChangeEventHandler([FrequencyParameters](#) *FrequencyArgs*)

Parameters: *FrequencyArgs*: Start Frequency, Stop Frequency, Center Frequency, and Frequency Span.

Structures

FrequencyParameters Structure

VB: Public Structure FrequencyParameters
C#: public struct FrequencyParameters
C++: public struct FrequencyParameters

Fields

CenterFrequency Field

Gets or sets the center frequency of the active remote spectrum monitor.

VB: Public CenterFrequency As [Long](#)
C#: public [long](#) CenterFrequency
C++: public [long](#) CenterFrequency

Property Value: Type: String. Units are expected. If no units are provided, Hz is assumed.

FrequencySpan Field

VB: Public FrequencySpan As [Long](#)
C#: public [long](#) FrequencySpan
C++: public [long](#) FrequencySpan

StartFrequency Field

The sweep Start Frequency in Hz.

VB: Public StartFrequency As [Long](#)

C#: public [long](#) StartFrequency

C++: public [long](#) StartFrequency

StopFrequency Field

The sweep Stop Frequency in Hz.

VB: Public StopFrequency As [Long](#)

C#: public [long](#) StopFrequency

C++: public [long](#) StopFrequency

Enumerations

BitDepth Enumeration

Designates the number of bits used in I/Q captures. Higher bit resolution increases the size of the download from each target receiver. Lower bit resolution is not appropriate for most purposes; the I/Q plot will look odd, with I/Q values lining up in rows and columns. The best choice for most applications is 16 bits.

Value:

Table 3-5. BitDepth Values and Definitions

Value	Definition
Eight	8-bits used for each I & Q value.
Ten	10-bits used for each I & Q value.
Sixteen	16-bits used for each I & Q value.
TwentyFour	24-bits used for each I & Q value.

GPSTDisplayMode Enumeration

Designates the display format for GPS coordinates.

Value:

Table 3-6. GPSTDisplayMode Values and Definitions

Value	Definition
GPS_Decimal	Sets GPS display format to decimal notation.
GPS_DMS	Sets GPS display format to Degrees-Minutes-Seconds.

Structures

FileHeader Structure

VB: Public Structure FileHeader

C#: public struct FileHeader

C++: public struct FileHeader

Fields

Bandwidth Field

The bandwidth of I/Q Capture.

VB: Public Bandwidth As Double

C#: public double Bandwidth

C++: public double Bandwidth

CaptureLength Field

Time in milliseconds of the I/Q Capture.

VB: Public CaptureLength As Integer

C#: public int CaptureLength

C++: public int CaptureLength

CenterFrequency Field

Gets or sets the center frequency of the active remote spectrum monitor.

VB: Public CenterFrequency As Double

C#: public double CenterFrequency

C++: public double CenterFrequency

Property Value: Type: String. Units are expected. If no units are provided, Hz is assumed.

TimeStamp Field

Sets the date and time.

VB: Public TimeStamp As Date

C#: public DateTime TimeStamp

C++: public DateTime TimeStamp

Version Field

Obtains the application version.

VB: Public Version As Integer

C#: public int Version

C++: public int Version

3-11 Scanner Class

Properties

Scanner.ChannelCount Property

Gets the number of channels in use in the remote spectrum monitor.

```
VB: Public ReadOnly Property ChannelCount() As Integer
C#: public int ChannelCount {get;}
C++: public int ChannelCount {get;}
```

Scanner.Connected Property

Gets the connection status of the remote spectrum monitor.

```
VB: Public ReadOnly Property Connected() As Boolean
C#: public bool Connected {get;}
C++: public bool Connected {get;}
```

Scanner.FileDelimiter Property

Gets or sets the file delimiter character used in the output file to separate fields in each record.

```
VB: Public Property FileDelimiter() As Delimiters
C#: public Delimiters FileDelimiter {get; set;}
C++: public Delimiters FileDelimiter {get; set;}
```

Scanner.FileViewer Property

Gets the File Viewer status or or sets the File Viewer to open the current CPM log file to view its contents.

```
VB: Public Property FileViewer() As ViewModes
C#: public ViewModes FileViewer {get; set;}
C++: public ViewModes FileViewer {get; set;}
```

Scanner.MaxFileSize Property

Gets or sets the maximum file size of the current CPM log file.

```
VB: Public Property MaxFileSize() As Long
C#: public long MaxFileSize {get; set;}
C++: public long MaxFileSize {get; set;}
```

Scanner.MinimumFreeSpace Property

Gets or sets the maximum free space of the current CPM log file.

```
VB: Public Property MinimumFreeSpace() As Integer
C#: public int MinimumFreeSpace {get; set;}
C++: public int MinimumFreeSpace {get; set;}
```

Scanner.ReloadCPC Property

Gets or sets the CPC file format for saving configuration files.

VB: Public Property ReloadCPC() As [Boolean](#)

C#: public [bool](#) ReloadCPC {get; set;}

C++: public [bool](#) ReloadCPC {get; set;}

Scanner.SaveGPS Property

Gets or sets the Save GPS status.

VB: Public Property SaveGPS() As [Boolean](#)

C#: public [bool](#) SaveGPS {get; set;}

C++: public [bool](#) SaveGPS {get; set;}

Scanner.SaveOnEvent Property

Gets the Save On Event status or sets Save On Event to Never, On limit violation, or Every Sweep.

VB: Public Property SaveOnEvent() As [SaveTrigger](#)

C#: public [SaveTrigger](#) SaveOnEvent {get; set;}

C++: public [SaveTrigger](#) SaveOnEvent {get; set;}

Scanner.SaveTraces Property

Gets the Save On Event status or sets Save On Event to Never, On limit violation, or Every Sweep.

VB: Public Property SaveTraces() As [Boolean](#)

C#: public [bool](#) SaveTraces {get; set;}

C++: public [bool](#) SaveTraces {get; set;}

Scanner.ScanDelay Property

Gets or sets the Scan Delay status.

VB: Public Property ScanDelay() As [Integer](#)

C#: public [int](#) ScanDelay {get; set;}

C++: public [int](#) ScanDelay {get; set;}

Scanner.ScanOnStartup Property

Gets or sets the option for running High-Speed Port Scanner automatically every time the computer is started.

VB: Public Property ScanOnStartup() As [Boolean](#)

C#: public [bool](#) ScanOnStartup {get; set;}

C++: public [bool](#) ScanOnStartup {get; set;}

Scanner.SetOutputFolder Property

Gets the Output Folder name or sets the Output Folder name.

VB: Public Property SetOutputFolder() As [String](#)

C#: public [string](#) SetOutputFolder {get; set;}

C++: public [string](#) SetOutputFolder {get; set;}

Scanner.ShowTraces Property

Gets or sets the Show Traces status.

```
VB: Public Property ShowTraces() As Boolean
C#: public bool ShowTraces {get; set;}
C++: public bool ShowTraces {get; set;}
```

Scanner.SweepInterval Property

Gets the current sweep interval or sets a new sweep interval.

```
VB: Public Property SweepInterval() As Integer
C#: public int SweepInterval {get; set;}
C++: public int SweepInterval {get; set;}
```

Methods

Scanner.AddChannel Method

Adds new channels to the list according to the parameters.

```
VB: Public Sub AddChannel(ByVal Channel As String)
C#: public System.Void AddChannel(string Channel)
C++: public System.Void AddChannel(string Channel)
```

Parameters: Channel: The number or index of the channel to add.

Scanner.AddChannelsDialog Method

Prompts the Add Channel Dialog to add another channel to the Channel List.

```
VB: Public Sub AddChannelsDialog()
C#: public System.Void AddChannelsDialog()
C++: public System.Void AddChannelsDialog()
```

Scanner.Close Method

The program closes the Scanner Add Channel control.

```
VB: Public Sub Close()
C#: public System.Void Close()
C++: public System.Void Close()
```

Scanner.ListChannels Method

The list of Channels of the remote spectrum monitor set in Parameter.s

```
VB: Public Function ListChannels(Optional ByVal Hostname As String = "") As String
C#: public string ListChannels(string Hostname = "")
C++: public string ListChannels(string Hostname = "")
```

Parameters: Hostname: Hostname or URL of the remote spectrum monitor.

Scanner.LoadConfiguration Method

Loads the configuration file set in Parameters.

VB: Public Function LoadConfiguration(ByVal *Filename* As String) As Boolean

C#: public bool LoadConfiguration(string *Filename*)

C++: public bool LoadConfiguration(string *Filename*)

Parameters: *Filename*: Filename, including path, where the scanner configuration is to be saved.

Scanner.Pause Method

Places channel monitoring in pause mode.

VB: Public Sub Pause()

C#: public System.Void Pause()

C++: public System.Void Pause()

Scanner.RemoveAllChannels Method

Removes all pre-existing channel definitions from the Channel List.

VB: Public Sub RemoveAllChannels()

C#: public System.Void RemoveAllChannels()

C++: public System.Void RemoveAllChannels()

Scanner.RemoveChannel Method

Removes a channel definition from the Channel List set in Parameters.

VB: Public Sub RemoveChannel(ByVal *ChannelIndex* As Integer)

C#: public System.Void RemoveChannel(int *ChannelIndex*)

C++: public System.Void RemoveChannel(int *ChannelIndex*)

Parameters: *ChannelIndex*: The channel number of the channel to be removed from the channel list.

Scanner.RunCounters Method

The number of times a channel has been scanned.

VB: Public Function RunCounters() As String

C#: public string RunCounters()

C++: public string RunCounters()

Scanner.RunSummary Method

Returns a string of elapse time, number of channels scanned, number of channels failed...

VB: Public Function RunSummary() As String

C#: public string RunSummary()

C++: public string RunSummary()

Scanner.SaveConfiguration Method

Saves the configuration file set in Parameters.

VB: Public Function SaveConfiguration(ByVal *Filename* As String) As Boolean

C#: public bool SaveConfiguration(string *Filename*)

C++: public bool SaveConfiguration(string *Filename*)

Parameters: *Filename*: The name of the configuration file to be saved.

Scanner.Start Method

Begins channel monitoring.

VB: Public Sub Start()

C#: public System.Void Start()

C++: public System.Void Start()

Events

ScanEvent Event

Scans the information for channel monitoring set in Parameters.

VB: Public Event ScanEvent(ByVal *Count* As Integer,ByVal *Rate* As Single,ByVal *Elapsed* As TimeSpan)

C#: public event ScanEventEventHandler ScanEvent
delegate void ScanEventEventHandler(int *Count*,float *Rate*,TimeSpan *Elapsed*)

C++: public event ScanEventEventHandler ScanEvent
delegate void ScanEventEventHandler(int *Count*,float *Rate*,TimeSpan *Elapsed*)

Parameters: *Count*: The total of scans through the channel list.

Rate: The channel per second captured and stored.

Elapsed: Elapse time of the of the current run.

Structures

MaskDef Structure

VB: Public Structure MaskDef

C#: public struct MaskDef

C++: public struct MaskDef

ScannerChannelDefinition Structure

VB: Public Structure ScannerChannelDefinition

C#: public struct ScannerChannelDefinition

C++: public struct ScannerChannelDefinition

Fields

Level Field

Power level per mask segment.

VB: Public Level As [Single](#)

C#: public [float](#) Level

C++: public [float](#) Level

StartFreq Field

The mask Start Frequency in Hz.

VB: Public StartFreq As [Double](#)

C#: public [double](#) StartFreq

C++: public [double](#) StartFreq

StopFreq Field

The mask Stop Frequency in Hz.

VB: Public StopFreq As [Double](#)

C#: public [double](#) StopFreq

C++: public [double](#) StopFreq

Antenna Field

Active antenna port number of multiport remote spectrum monitor.

VB: Public Antenna As [Integer](#)

C#: public [int](#) Antenna

C++: public [int](#) Antenna

ChannelPower Field

Returns the Channel Power calculated for the most recent trace acquired from the active remote spectrum monitor.

VB: Public ChannelPower As [Single](#)

C#: public [float](#) ChannelPower

C++: public [float](#) ChannelPower

DeltaMarker Field

Inputs the frequency of the delta marker relative to the marker position.

VB: Public DeltaMarker As [Single](#)

C#: public [float](#) DeltaMarker

C++: public [float](#) DeltaMarker

DeltaPower Field

Returns power at the delta marker position.

VB: Public DeltaPower As [Single](#)

C#: public [float](#) DeltaPower

C++: public [float](#) DeltaPower

Description Field

Any information to describe the channel by the user.

VB: Public Description As [String](#)

C#: public [string](#) Description

C++: public [string](#) Description

DisplayPoints Field

The number of points in the trace.

VB: Public DisplayPoints As [Integer](#)

C#: public [int](#) DisplayPoints

C++: public [int](#) DisplayPoints

Hostname Field

Enter the hostname or URL address for the instrument taking measurements.

VB: Public Hostname As [String](#)

C#: public [string](#) Hostname

C++: public [string](#) Hostname

ListIndex Field

The specific number of the channel in the list.

VB: Public ListIndex As [Integer](#)

C#: public [int](#) ListIndex

C++: public [int](#) ListIndex

LowerLimit Field

A power value of the lower limit.

VB: Public LowerLimit As [Single](#)

C#: public [float](#) LowerLimit

C++: public [float](#) LowerLimit

LowerMask Field

The mask lower limit.

VB: Public LowerMask As [String](#)

C#: public [string](#) LowerMask

C++: public [string](#) LowerMask

Marker Field

The frequency position of a marker.

VB: Public Marker As [Single](#)

C#: public [float](#) Marker

C++: public [float](#) Marker

MarkerPower Field

The power at the marker position.

VB: Public MarkerPower As [Single](#)

C#: public [float](#) MarkerPower

C++: public [float](#) MarkerPower

OBW Field

Enter the Occupied Bandwidth for the most recent trace acquired from the active target receiver.

VB: Public OBW As [Single](#)

C#: public [float](#) OBW

C++: public [float](#) OBW

PassRate Field

The total number of traces in the channels, the number or traces that passed, and the pass percentage.

VB: Public PassRate As [Single](#)

C#: public [float](#) PassRate

C++: public [float](#) PassRate

PassStatus Field

Pass/Fail status when trace compared to mask.

VB: Public PassStatus As [Boolean](#)

C#: public [bool](#) PassStatus

C++: public [bool](#) PassStatus

Preamp Field

Sets the Preamp field for measurement configuration.

VB: Public Preamp As [Boolean](#)

C#: public [bool](#) Preamp

C++: public [bool](#) Preamp

RBW Field

The Resolution Bandwidth setting for the channel.

VB: Public RBW As [Double](#)

C#: public [double](#) RBW

C++: public [double](#) RBW

RefLevel Field

The Reference Level in dBm for the channel.

VB: Public RefLevel As [Single](#)

C#: public [float](#) RefLevel

C++: public [float](#) RefLevel

StartFreq Field

The sweep Start Frequency in Hz.

VB: Public StartFreq As [Double](#)

C#: public [double](#) StartFreq

C++: public [double](#) StartFreq

StopFreq Field

The sweep Stop Frequency in Hz.

VB: Public StopFreq As [Double](#)

C#: public [double](#) StopFreq

C++: public [double](#) StopFreq

TraceCount Field

The trace sweep count useful for multiple-sweep measurements.

VB: Public TraceCount As [Integer](#)

C#: public [int](#) TraceCount

C++: public [int](#) TraceCount

UpperLimit Field

Upper power limit.

VB: Public UpperLimit As [Single](#)

C#: public [float](#) UpperLimit

C++: public [float](#) UpperLimit

UpperMask Field

Upper mask definition.

VB: Public UpperMask As [String](#)

C#: public [string](#) UpperMask

C++: public [string](#) UpperMask

VBW Field

The Video bandwidth setting for the channel.

VB: Public VBW As [Double](#)

C#: public [double](#) VBW

C++: public [double](#) VBW

strTrace Field

Comma delimited string of trace data.

VB: Public strTrace As [String](#)

C#: public [string](#) strTrace

C++: public [string](#) strTrace

Enumerations

Delimiters Enumeration

Designates the column delimiter used to separate values in CPM files.

Value:

Table 3-7. Delimiters Values and Definitions

Value	Definition
Comma	Comma
Semicolon	Semi-colon
Tab	Tab character
Space	Space character

SaveModes Enumeration

Designates the file mode for the High-Speed Port Scanner.

Value:

Table 3-8. SaveModes Values and Definitions

Value	Definition
NewFile	Create a new file each time scanning starts.
Append	Append new traces to the current file when scanning starts.

SaveTrigger Enumeration

Designates the trigger (event) that causes traces in the High-Speed Port Scanner to be written to the CPM file.

Value:

Table 3-9. SaveTrigger Values and Definitions

Value	Definition
None	Nothing is written to disk file. This mode is for visual monitoring on screen.
FailsOnly	Only write results for sweeps that fail a designated mask or threshold power level.
AllResults	Write sweep results to the CPM file after each sweep. This causes very large files, especially if trace data is being saved. This is the preferred mode if the goal is to do post-processing to look for events, coverage mapping, occupancy, etc.

ViewModes Enumeration

Description: The High-Speed Port Scanner has a built-in CPM file viewer with a very basic display. Vision also includes a File Viewer specifically for looking at CPM and RSM data files. The external option provides a much richer experience. There are circumstances where the High-Speed Port Scanner is deployed as a stand-alone program without the other Vision components available. It is recommended to use the Vision File Viewer (external) when available.

Table 3-10. ViewModes Values and Definitions

Value	Definition
Internal	High-Speed Port Scanner uses its own simple file viewer to display CPM file contents.
External	High-Speed Port Scanner uses the Vision File Viewer to open and display CPM file contents.

3-12 TDOAControl Class

Properties

TDOAControl.AntennaChannel Property

Gets or sets the Antenna Channel set in Parameters for making TDOA measurements.

VB: Public Property AntennaChannel(ByVal *Index* As Integer) As Integer

C#: [System.Runtime.CompilerServices.IndexerName("AntennaChannel")]
public int this[int *Index*] {get; set;}

C++: [System.Runtime.CompilerServices.IndexerName("AntennaChannel")]
public int this[int *Index*] {get; set;}

Parameters: *Index*: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.AutoReferenceLevels Property

Gets or sets the reference level amplitude value.

VB: Public Property AutoReferenceLevels() As Boolean

C#: public bool AutoReferenceLevels {get; set;}

C++: public bool AutoReferenceLevels {get; set;}

TDOAControl.AverageCount Property

Gets or sets the average count for use in calculating the average display value.

VB: Public Property AverageCount() As Integer

C#: public int AverageCount {get; set;}

C++: public int AverageCount {get; set;}

TDOAControl.AverageDistances Property

Gets or sets averaging for TDOA measurement results.

VB: Public Property AverageDistances() As Boolean

C#: public bool AverageDistances {get; set;}

C++: public bool AverageDistances {get; set;}

TDOAControl.CaptureBandwidth Property

Gets or sets the I/Q Capture data bandwidth.

VB: Public Property CaptureBandwidth() As String

C#: public string CaptureBandwidth {get; set;}

C++: public string CaptureBandwidth {get; set;}

TDOAControl.CenterFrequency Property

Gets or sets the center frequency of the active target receiver.

VB: Public Property CenterFrequency() As [String](#)

C#: public [string](#) CenterFrequency {get; set;}

C++: public [string](#) CenterFrequency {get; set;}

Property Value: Type: String. Units are expected. If no units are provided, Hz is assumed.

TDOAControl.DrawEstimateLines Property

Gets or sets status to draw estimate lines on the map.

VB: Public Property DrawEstimateLines() As [Boolean](#)

C#: public [bool](#) DrawEstimateLines {get; set;}

C++: public [bool](#) DrawEstimateLines {get; set;}

TDOAControl.DrawHeatMap Property

Gets or sets status to draw a heat map.

VB: Public Property DrawHeatMap() As [Boolean](#)

C#: public [bool](#) DrawHeatMap {get; set;}

C++: public [bool](#) DrawHeatMap {get; set;}

TDOAControl.DrawSweepTrace Property

Gets or sets status to draw graph on screen.

VB: Public Property DrawSweepTrace() As [Boolean](#)

C#: public [bool](#) DrawSweepTrace {get; set;}

C++: public [bool](#) DrawSweepTrace {get; set;}

TDOAControl.GraphMode Property

Gets the current Graph Mode or sets the Graph Mode to either 1 Probe, 3 Probes, or TDOA.

VB: Public Property GraphMode() As [ChartType](#)

C#: public [ChartType](#) GraphMode {get; set;}

C++: public [ChartType](#) GraphMode {get; set;}

TDOAControl.Latitude Property

The obtains the latitude coordinate of the remote spectrum monitor.

VB: Public ReadOnly Property Latitude() As [Single](#)

C#: public [float](#) Latitude {get;}

C++: public [float](#) Latitude {get;}

TDOAControl.Longitude Property

The longitude coordinate of the remote spectrum monitor.

VB: Public ReadOnly Property Longitude() As [Single](#)

C#: public [float](#) Longitude {get;}

C++: public [float](#) Longitude {get;}

TDOAControl.MovingAverageCount Property

Gets or sets the moving average count for use in calculating the average display value.

```
VB: Public Property MovingAverageCount() As Integer
C#: public int MovingAverageCount {get; set;}
C++: public int MovingAverageCount {get; set;}
```

TDOAControl.Preamp Property

Sets the Preamp parameter for measurement configuration.

```
VB: Public Property Preamp(ByVal Index As Integer) As Boolean
C#: [System.Runtime.CompilerServices.IndexerName("Preamp")]
    public bool this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("Preamp")]
    public bool this[int Index] {get; set;}
```

Parameters: Index: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.ProbeDelay Property

Gets the current Probe Delay value or sets the value set by Parameters.

```
VB: Public Property ProbeDelay(ByVal Index As Integer) As Integer
C#: [System.Runtime.CompilerServices.IndexerName("ProbeDelay")]
    public int this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("ProbeDelay")]
    public int this[int Index] {get; set;}
```

Parameters: Index: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.ProbeHostname Property

Enter the hostname or URL address for the instrument taking measurements.

```
VB: Public Property ProbeHostname(ByVal Index As Integer) As String
C#: [System.Runtime.CompilerServices.IndexerName("ProbeHostname")]
    public string this[int Index] {get; set;}
C++: [System.Runtime.CompilerServices.IndexerName("ProbeHostname")]
    public string this[int Index] {get; set;}
```

Parameters: Index: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.ProbeStatus Property

Returns status of probe. True if probe is online.

```
VB: Public ReadOnly Property ProbeStatus(ByVal Index As Integer) As Boolean
C#: [System.Runtime.CompilerServices.IndexerName("ProbeStatus")]
    public bool this[int Index] {get;}
C++: [System.Runtime.CompilerServices.IndexerName("ProbeStatus")]
    public bool this[int Index] {get;}
```

Parameters: Index: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.ReferenceLevel Property

Gets or sets the reference level amplitude value.

VB: Public Property ReferenceLevel(ByVal *Index* As Integer) As Integer

C#: [System.Runtime.CompilerServices.IndexerName("ReferenceLevel")]
public int this[int *Index*] {get; set;}

C++: [System.Runtime.CompilerServices.IndexerName("ReferenceLevel")]
public int this[int *Index*] {get; set;}

Parameters: *Index*: The TDOA measurement uses 3 RSM probes for capturing I/Q data and calculating the location estimate. The probes are indexed, 0, 1, and 2.

TDOAControl.ReportFolder Property

Gets or sets the name of the TDOA Report.

VB: Public Property ReportFolder() As String

C#: public string ReportFolder {get; set;}

C++: public string ReportFolder {get; set;}

TDOAControl.RunCount Property

Gets the number of used in calculated the average.

VB: Public ReadOnly Property RunCount() As Integer

C#: public int RunCount {get;}

C++: public int RunCount {get;}

TDOAControl.ShowCorrelationResults Property

Gets or sets the correlation results.

VB: Public Property ShowCorrelationResults() As Boolean

C#: public bool ShowCorrelationResults {get; set;}

C++: public bool ShowCorrelationResults {get; set;}

TDOAControl.ShowGraphs Property

Gets or sets the status of ShowGraphs.

VB: Public Property ShowGraphs() As Boolean

C#: public bool ShowGraphs {get; set;}

C++: public bool ShowGraphs {get; set;}

Property Value: **true** if yes; otherwise, **false**.

TDOAControl.ShowHelp Property

Gets or sets the status of ShowHelp. Help is displayed if status is “true”.

VB: Public Property ShowHelp() As Boolean

C#: public bool ShowHelp {get; set;}

C++: public bool ShowHelp {get; set;}

TDOAControl.ShowMap Property

Gets or sets the status of Show Map.

```
VB: Public Property ShowMap() As Boolean
C#: public bool ShowMap {get; set;}
C++: public bool ShowMap {get; set;}
```

TDOAControl.ShowMobileTracking Property

Gets or sets the status of Show Mobile Tracking.

```
VB: Public Property ShowMobileTracking() As Boolean
C#: public bool ShowMobileTracking {get; set;}
C++: public bool ShowMobileTracking {get; set;}
```

TDOAControl.ShowViewOptions Property

Gets or sets the status of Show View Options.

```
VB: Public Property ShowViewOptions() As Boolean
C#: public bool ShowViewOptions {get; set;}
C++: public bool ShowViewOptions {get; set;}
```

TDOAControl.TrackMovingTarget Property

Gets or sets the status of Track Moving Target.

```
VB: Public Property TrackMovingTarget() As Boolean
C#: public bool TrackMovingTarget {get; set;}
C++: public bool TrackMovingTarget {get; set;}
```

TDOAControl.WaitForPeak Property

Gets or sets the status of Wait For Peak.

```
VB: Public Property WaitForPeak() As Boolean
C#: public bool WaitForPeak {get; set;}
C++: public bool WaitForPeak {get; set;}
```

Methods**TDOAControl.GenerateReport Method**

Activates the program to generate a TDOA report.

```
VB: Public Sub GenerateReport()
C#: public System.Void GenerateReport()
C++: public System.Void GenerateReport()
```

TDOAControl.GetParametersFromInput Method

Opens dialog input for I/Q Capture parameters.

```
VB: Public Sub GetParametersFromInput()  
C#: public System.Void GetParametersFromInput()  
C++: public System.Void GetParametersFromInput()
```

TDOAControl.ResetAveraging Method

Enter the channel number of the active remote spectrum monitor.

```
VB: Public Sub ResetAveraging()  
C#: public System.Void ResetAveraging()  
C++: public System.Void ResetAveraging()
```

TDOAControl.RunTDOA Method

Begins the TDOA application.

```
VB: Public Sub RunTDOA()  
C#: public System.Void RunTDOA()  
C++: public System.Void RunTDOA()
```

TDOAControl.SaveIQData Method

Saves the IQData to a file set in Parameter.

```
VB: Public Sub SaveIQData(Optional ByVal Filename As String = "")  
C#: public System.Void SaveIQData(string Filename = "")  
C++: public System.Void SaveIQData(string Filename = "")
```

Parameters: *Filename*: The name of the file that the IQData is saved.

TDOAControl.ScreenShot Method

Triggers a screen capture and places it into the clipboard.

```
VB: Public Function ScreenShot() As Bitmap  
C#: public Bitmap ScreenShot()  
C++: public Bitmap ScreenShot()
```

TDOAControl.VerifyProbe Method

Verifies that a probe is available and ready set by Parameters.

```
VB: Public Function VerifyProbe(ByVal Number As Integer) As Boolean  
C#: public bool VerifyProbe(int Number)  
C++: public bool VerifyProbe(int Number)
```

Parameters: *Number*: The number of the probe to be verified.

Events

CalibrateProbes Event

VB: Public Event CalibrateProbes()
C#: public event CalibrateProbesEventHandler CalibrateProbes
delegate void CalibrateProbesEventHandler()
C++: public event CalibrateProbesEventHandler CalibrateProbes
delegate void CalibrateProbesEventHandler()

TDOA_Iteration Event

Returns notification that a single TDOA measurement has been completed.

VB: Public Event TDOA_Iteration()
C#: public event TDOA_IterationEventHandler TDOA_Iteration
delegate void TDOA_IterationEventHandler()
C++: public event TDOA_IterationEventHandler TDOA_Iteration
delegate void TDOA_IterationEventHandler()

Structures

ChannelInfo Structure

VB: Public Structure ChannelInfo
C#: public struct ChannelInfo
C++: public struct ChannelInfo

Fields

Channel Field

Enter the channel number of the active remote spectrum monitor.

VB: Public Channel As Integer
C#: public int Channel
C++: public int Channel

Latitude Field

The latitude coordinate of the remote spectrum monitor.

VB: Public Latitude As Double
C#: public double Latitude
C++: public double Latitude

Longitude Field

The longitude coordinate of the remote spectrum monitor.

VB: Public Longitude As Double
C#: public double Longitude
C++: public double Longitude

ProbeName Field

Enter a name for the desired Remote Spectrum Analyzer.

```
VB: Public ProbeName As String
C#: public string ProbeName
C++: public string ProbeName
```

Ready Field

Checks status that the prober is ready for measurement.

```
VB: Public Ready As Boolean
C#: public bool Ready
C++: public bool Ready
```

URL Field

Enter a URL for the desired Remote Spectrum Analyzer.

```
VB: Public URL As String
C#: public string URL
C++: public string URL
```

Enumerations

ChartType Enumeration

Designates the Chart type to be displayed in the TDOA control. If Sweep is selected, then a trace sweep is acquired and displayed before each I/Q capture occurs. This allows the user to monitor the RF signal to verify that the component of interest is still there.

Value:

Table 3-11. ChartType Values and Definitions

Value	Definition
IQ	Display an I/Q plot with each TDOA measurement iteration.
Sweep	Display a sweep trace with each TDOA measurement iteration.

3-13 Trace Class

Trace.StartFrequency()

Sets/Gets the start frequency for trace data passed in with the TraceString property. This is necessary to produce meaningful output in the channel list. The input is a text value that should contain units. For example: 743.35 MHz

```
VB: Public Property StartFrequency() As String
C#: public string StartFrequency {get; set;}
C++: public string StartFrequency {get; set;}
```

Trace.StopFrequency()

Sets/Gets the stop frequency for trace data passed in with the TraceString property. This is necessary to produce meaningful output in the channel list. The input is a text value that should contain units. For example: 743.35 MHz

VB: Public Property StopFrequency() As [String](#)

C#: public [string](#) StopFrequency {get; set;}

C++: public [string](#) StopFrequency {get; set;}

Trace.RBW()

Sets/Gets the resolution bandwidth for trace data passed in with the TraceString property. This is necessary to produce meaningful output in the channel list. The input is a text value that should contain units. For example: 100 kHz

VB: Public Property RBW() As [String](#)

C#: public [string](#) RBW {get; set;}

C++: public [string](#) RBW {get; set;}

Trace.dbDown()

Sets/Gets the value used to determine channels in the trace data. A peak that is dbDown above trace points on either side will constitute a channel. A typical value might be 9 dB, or 20 dB. This value is not entered with units.

VB: Public Property dbDown() As [String](#)

C#: public [string](#) dbDown {get; set;}

C++: public [string](#) dbDown {get; set;}

Trace.TraceString()

Sets the trace data as a comma delimited string of power values. This string can come from many sources, but typically will come either from using the RSM.Trace property or the Monitor.TraceData property

VB: Public Property TraceString() As [String](#)

C#: public [string](#) TraceString {get; set;}

C++: public [string](#) TraceString {get; set;}

Trace.Power(int Index)

(Read Only) Returns the power (amplitude) of the trace at the designated index point. The index is the ordinal position of the power value. This does not accept frequency values. You will need to calculate an index into the power array from the frequency you desire.

VB: Public Property Power(int Index)(ByVal *Index* As [Integer](#)) As [Single](#)

C#: [System.Runtime.CompilerServices.IndexerName("Power(int Index)")]
public float this[int *Index*] {get; set;}

C++: [System.Runtime.CompilerServices.IndexerName("Power(int Index)")]
public float this[int *Index*] {get; set;}

Trace.SortChannels(channelSort Sortby)

Sets the sort order for the Channel list. Input value is one of:

Trace.ChannelSort.CenterFrequency
Trace.ChannelSort.PeakFrequency
Trace.ChannelSort.PeakPower
Trace.ChannelSort.ChannelPower
Trace.ChannelSort.ChannelWidth

VB: Public Sub SortChannels(ByVal Channelsort As [Sortby](#))

C#: Public [System.Void](#) SortChannels([Sortby](#) Channelsort)

C++: Public [System.Void](#) SortChannels([Sortby](#) Channelsort)

Trace.Channels()

Returns the channel list discovered in the trace data. You must first set the above parameters before this produces the desired result.

Trace.Channels is a list of type Channel Definition. This has the following members.

CenterFrequency
PeakFrequency
PeakPower
ChannelPower
ChannelWidth

VB: Public ReadOnly Property Channels() As [ChannelDefinition](#)

C#: Public [ChannelDefinition](#) Channels {get;}

C++: Public [ChannelDefinition](#) Channels {get;}



10580-00435



D



Anritsu utilizes recycled paper and environmentally conscious inks and toner.

Anritsu Company
490 Jarvis Drive
Morgan Hill, CA 95037-2809
USA
<http://www.anritsu.com>