Programming Information

(Printed Version of Help)

Agilent Technologies
PNA Series Network Analyzers

Agilent Technologies

Part Number: E8356-90028
Printed in USA
January 2003

Supersedes December 2002

COM versus SCPI

There are two methods you can use to remotely control the analyzer: COM and SCPI. The following topics are intended to help you choose the method that best meets your needs:

- Software Connection
- Physical Connection
- Selecting a Method
- Programming Languages

Other Topics about COM Concepts

Software Connection

COM uses a binary protocol, allowing the user to directly invoke a feature of the Network Analyzer. This is more efficient than SCPI. For example, the following statement calls directly into the Network Analyzer, executing the routine GetIDString.

```
PNA.GetIDString()
```

SCPI is a text based instrument language. To retrieve the ID string, you would send the following text string to the network analyzer:

```
IbWrite( "*IDN?")
```

The network analyzer's SCPI parser would first decode this text string to determine that the user has asked for the network analyzer to identify itself. Then the parser would call GetIDString().

The Physical Connection

Internal Control

With either COM or SCPI, the best throughput is attained by using the analyzer's internal PC to execute your test code. However, if your test code uses too much system resources (CPU cycles and/or memory), this will slow the Analyzer's performance.

Using the SICL I/O Libraries, you can also connect to the Analyzer from a program running on the Analyzer.

External Control

You can control the analyzer from a remote PC using either COM or SCPI.

COM - (Component Object Model) can be used to access any program like the analyzer (835x.exe) or library (.dll) that exposes its features using a COM compliant object model. These programs or libraries are called "servers". Programs (like your remote program on your PC) that connect to and use the features of these servers are called "clients."

With COM, the server and the client do not need to reside on the same machine. DCOM, or distributed COM, is easy to configure and makes the location of the server transparent to the client. When you access the Analyzer from a remote computer, you are using DCOM. In this case, the mechanical transport is a LAN (local area network).

SCPI - Using a GPIB interface card in a remote computer, you can connect to the instrument using a GPIB cable. There are some constraints on the length of this cable and the number of instruments that can be daisy-chained together.

Using the Agilent SICL I/O libraries, you can connect to the instrument over a LAN connection.

(LAN or INTERNAL) You can send SCPI commands using COM with an object called the ScpiStringParser. This object provides access to the SCPI parser (or command decoder) so that you can send SCPI text commands using automation.

Selecting a Method

You should almost always choose COM for the following reasons:

- COM executes faster most of the time.
- COM is generally easier to use. The latest development tools embrace COM and know how to make your life easier with integrated development environments that show automation syntax as you type.
- As time goes on, more emphasis will be put on the COM as the preferred programming paradigm. As new capability is developed, it may not be made available through SCPI.

But choosing a connection method depends on your situation. Here are some additional things to consider:

1. If you want to use the Analyzer to control other GPIB instruments, you may want to use COM as the means of talking to the instrument. In GPIB, the analyzer can not be configured as both **System Controller** and **talker/listener**. Because the Analyzer does not support pass control mode, only one mode can be used at a time.
2. If you have legacy code written in SCPI for another network analyzer, you may be able to leverage that code to control the Analyzer. However, the PNA uses a different platform than previous Agilent Network Analyzers. Therefore, not all commands have a direct replacement. See the 8753 command finder.

Programming Languages

You can program the Analyzer with either COM or SCPI using several languages. The most common include:

Agilent VEE - With this language you can send text based SCPI commands and also use automation. VEE 6.0 or later is recommended.

Visual Basic - This language has great support for automation objects and can be used to drive SCPI commands. The use of VISA drivers for your GPIB hardware interface will make the task of sending SCPI commands easier.

C++ - This language can do it all. It is not as easy to use as the above two, but more flexible.

Command Finder

File Commands

Description	SCPI	COM
Save States (Inst I Cal I Both)	MMEMory:STORe	
Recall States (Inst ICal I Both)	MMEMory:LOAD	app.recall
Manage Files		
List Files	MMEMory:CATalog	
Copy Files	MMEMory:COPY	
Move Files	MMEMory:MOVE	
Delete Files	MMEMory:DELete	
Manage Folders		
Change	MMEMory:CDIRectory	
Delete	MMEMory:RDIRectory	
Make	MMEMory:MDIRectory	

Print		
Print	HCOPy	app.DoPrint
Print to File		app.PrintToFile

View Commands

Description	SCPI	COM
Status Bar On|Off	DISP:ANN:STAT	app.ShowStatusBar
Toolbars On|Off		app.ShowToolbar
Tables On|Off	DISP:WIND:TABLe	win.ShowTable
Title Bars On|Off		app.ShowTitleBars
X-axis values On|Off	DISP:ANN:FREQ	app.ShowStimulus
Marker Readout On|Off	DISP:WIND:ANN:MARK:STAT	win.MarkerReadout
One Readout per Trace	DISP:WIND:ANN:MARK:SING	win.OneReadoutPerTrace
Marker Readout Size	DISP:WIND:ANN:MARK:SIZE	win.MarkerReadoutSize
Measurement Trace On|Off	DISP:WIND:TRAC	meas.View
Memory Trace On|Off	DISP:WIND:TRAC:MEM	meas.View
Title Annotation On|Off	DISP:WIND:TITL	win.TitleState
Make a Title Annotation	DISP:WIND:TITL:DATA	win.Title
Display Update On|Off	DISP:ENAB	
Window Update On|Off	DISP:WIND:ENABle	
Analyzer Visible On|Off		app.Visible
Add a Window		wins.Add
Return a Window Number		win.WindowNumber
Activate a Window		app.ActivateWindow
Arrange Measurement Windows		app.ArrangeWindows
Analyzer Window (Max |Min| Normal)		app.WindowState
Display Automation Errors		app.DisplayAutomationErrors

Channel Commands

Power | Average | Offset | Manage

Description	SCPI	COM
Preset		app.Preset
Start Freq	SENS:FREQ:STAR	chan.StartFrequency
Stop Freq	SENS:FREQ:STOP	chan.StopFrequency
Center Freq	SENS:FREQ:CENT	chan.CenterFrequency
Span	SENS:FREQ:SPAN	chan.FrequencySpan
CW Frequency	SENS:FREQ:CW	chan.CWFrequency
Power Settings		
Power ON | OFF	OUTP	app.SourcePowerState

Power Value	SOUR:POW1	chan.TestPortPower
Port Selection	SENS:SWE:SRCP	chan.TestPortPower
Couple Ports OFF I ON	SOUR:POW:COUP	chan.CouplePorts
Attenuator Mode Auto|Manual	SOUR:POW:ATT:Auto	chan.Attenuator
Attenuation Value	SOUR:POW:ATT	chan.AttenuatorMode
Power Slope Mode Manual I Auto	SOUR:POW:SLOP:STAT	app.PowerSlope
Power Slope Value	SOUR:POW:SLOP	app.PowerSlope
Receiver Attenuation	SENS:POW:ATT	chan.ReceiverAttenuator
Averaging		
Average ON|OFF	SENS:AVER	chan.Average
Average Factor	SENS:AVER:COUN	chan.AveragingFactor
Return the Average Count		chan.AveragingCount
Average Restart	SENS:AVER:CLE	chan.AveragingRestart
Frequency Offset		
Offset mode ON I OFF	SENS:OFFS:STAT	chan.FrequencyOffsetState
Offset Frequency	SENS:OFFS:OFFS	chan.FrequencyOffsetFrequency
Read Offset Start Frequency	SENS:OFFS:STAR?	
Read Offset Stop Frequency	SENS:OFFS:STOP?	
Set Offset Multiplier	SENS:OFFS:MULT	chan.FrequencyOffsetMultiplier
Set Offset Divisor	SENS:OFFS:DIV	chan.FrequencyOffsetDivisor
Set CW Override	SENS:OFFS:CW	chan.FrequencyOffsetCWOverride
Test Set Switch	ROUT:PATH:LOOP:R1	chan.R1InputPath
Manage Channels		
Add		chans.Add
Make Active		app.ActiveChannel
Read Channel Number		chan.ChannelNumber
Read Number of Channels		chans.Count
Set up a copy of the Channel		chan.CopyToChannel

Sweep Commands

Power I Segment I Trigger

Description	SCPI	COM
Sweep Time Value	SENS:SWE:TIME:AUTO	chan.centerFrequency
IF Bandwidth	SENS:BWID	chan.IFBandwidth
Previous IF Bandwidth		chan.Previous_IFBandwidth
Next IFBandwidth		chan.Next_IFBandwidth
Number of Points	SENS:SWE:POIN	chan.NumberOfPoints
Sweep Type (Lin I Pwr I CW I Seg)	SENS:SWE:TYPE	chan.SweepType
Sweep Generation (Stepped I Analog)	SENS:SWE:GEN	chan.SweepGenerationMode
Dwell Time Value	SENS:SWE:DWEL	chan.DwellTime
Alternate Sweeps	SENS:COUP	chan.AlternateSweep

External ALC	SOUR:POW:DET	app.ExternalALC
Power Sweep		
Start Power	SOUR:POW:STAR	chan.StartPower
Stop Power	SOUR:POW:STOP	chan.StopPower
Center	SOUR:POW:CENT	
Span	SOUR:POW:SPAN	
Segment Sweep		
ON|OFF	SENS:SEGM	Seg.State
Add a segment	SENS:SEGM:ADD	Segs.Add
Delete a segment	SENS:SEGM:DEL	segments.Remove
Delete all segments	SENS:SEGM:DEL:ALL	
Count the segments	SENS:SEGM:COUN	chans.Count
Read the segment number		seg.SegmentNumber
Segment Center Frequency	SENS:SEGM:FREQ:CENT	chan.centerFrequency
Segment Frequency Span	SENS:SEGM:FREQ:SPAN	chan.FrequencySpan
Segment Start Frequency	SENS:SEGM:FREQ:STAR	Chan.StartFrequency
Segment Stop Frequency	SENS:SEGM:FREQ:STOP	Chan.StopFrequency
Number of Points	SENS:SEGM:SWE:POIN	seg.NumberOfPoints
IF Bandwidth	SENS:SEGM:BWID	seg.IFBandwidth
IF Bandwidth Option	SENS:SEGM:BWID:CONT	segs.IFBandwidthOption
Source Power	SENS:SEGM:POW	chan.TestPortPower
Source Power Option	SENS:SEGM:POW:CONT	segs.SourcePowerOption
X-Axis Point Spacing	SENS:SEGM:X:SPAC	chan.XAxisPointSpacing
Upload a segment table		SetAllSegmernts
Trigger		
Source (where trigger comes from)		
Trigger Source (Int | Ext | Manual)	TRIG:SOUR	app.TriggerSignal
Internal | Manual	INIT:CONT	
Trigger! (for Manual Source)	INIT	app.ManualTrigger
Ext. Trigger Slope (Positive | Negative)	TRIG:LEV	app.TriggerSignal
Trigger Delay	TRIG:DEL	app.TriggerDelay
Scope (what is triggered)		
Trigger Scope (Global | Channel)	TRIG:SCOP	app.TriggerType
Channel Settings (how the channel responds to triggers)		
Cont | Groups | Hold	SENS:SWE:MODE	
Continuous		chan.Continuous
Number of Groups	SENS:SWE:GRO:COUN	chan.NumberOfGroups
Hold		chan.Hold
Single		chan.Single
Trigger Mode (Point | Measurement)	SENS:SWE:TRIG:POIN	chan.TriggerMode
Restart	INIT	
Abort	ABOR	chan.Abort

Calibrate Commands

Guided | ECAL | Save-Recall | Cal Sets | CORR | Modify Kits | Standards | Power Cal | Cal Data

Description	SCPI	COM
Perform an Unguided Calibration		
Launch Cal Wizard	SYSTem:CORR:WIZard	app.LaunchCalWizard
Set Cal Type	SENS:CORR:COLL:METHod	cal.SetCalInfo
Select a Cal Kit	SENS:CORR:COLLect:CKIT	app.CalKitType
Get a Handle to the Active Cal Kit		app.ActiveCalKit
Simultaneous 2-Port Calibration	SENS:CORR:TSTandards	cal.Simultaneous2PortAcquisition
Acquisition Direction	SENS::CORR:SFORward	cal.AcquisitionDirection
Measure a Standard	SENS:CORR:COLLect	cal.AcquireCalStandard
Calculate Errors	SENS:CORR:COLL:SAVE	cal.CalculateErrorCoeffecients
Isolation ON|OFF	SENS:CORR:ISOLation	cal.AcquireCalStandard
Perform a Guided Cal		
Initiate a Guided Cal	SENS:CORR:COLL:GUID:INIT	
List valid Connector Types for a Port	SENS:CORR:COLL:GUID:CONN:CAT?	
List valid Cal Kits for a Port	SENS:CORR:COLL:GUID:CKIT:PORT:CAT?	
Select a Connector Type	SENS:CORR:COLL:GUID:CONN:PORT	
Select a Cal Kit	SENS:CORR:COLL:GUID:CKIT:PORT	
Return Number of Steps in a Cal	SENS:CORR:COLL:GUID:STEPs?	
Return a Description of a Cal Step	SENS:CORR:COLL:GUID:DESC?	
Measure a Cal Standard in a Guided Cal	SENS:CORR:COLL:GUID:ACQuire	
Calculate Error Terms from a Guided Cal	SENS:CORR:COLL:GUID:SAVE	
Perform an ECAL		
Do ECAL 1-Port	SENS:CORR:COLL:CKIT 99	cal.DoECAL1Port
Do ECAL 2-Port	SENS:CORR:COLL:CKIT 99	cal.DoECAL2Port
Get ECAL Module Info	SENS:CORR:COLL:CKIT:INF?	cal.GetECALModuleInfo
Confidence Check Parameter	SENS:CORR:CCH:PAR	
Confidence Check Acquire	SENS:CORR:CCHeck	cal.AcquireCalConfidenceCheckECAL
Confidence Check Done	SENS:CORR:CCH:DONE	cal.DoneCalConfidenceCheckECAL
Maps ECAL Module to PNA Ports	SENS:CORR:PREF:ECAL:PMAP	cal.ECALPortMap
Perform Module Orientation during calibration	SENS:CORR:PREF:ECAL:ORI	cal.OrientECALModule
Recall / Save / Apply a Calibration or Error Term		
Recall a Calibration	SENS:CORR:CSET	app.Recall
Apply a Calibration to a measurement	SENS:CORR:CSET	
Save a Calibration	SENS:CORR:CSET:SAVE	app.Save
Save or Recall an Error Term	CALC:DATA Scorr	Data Topic
Apply an Error Term after Uploading	SENS:CORR:COLLect:APPLy	
Cal Sets		
Create a Cal Set		calMgr.CreateCalSet
Delete a Cal Set	SENS:CORR:CSET:DEL	calMgr.DeleteCalSet
List Cal Sets	SENS:CORR:CSET:CAT?	calMgr.GetCalSetCatalog

Get Cal Set Information		calMgr.GetCalSetUsageInfo
Select a Cal Set by GUID	SENS:CORR:CSET:GUID	calMgr.GetCalSetByGUID
Select a Cal Set from a channel		channel.SelectCalSet
Copy a Cal Set		CalSet.Copy
Save a Cal Set		CalSet.Save
Save Cal Sets	SENS:CORR:CSET:SAVE	app.SaveCalSets
Change the Description of a Cal Set	SENS:CORR:CSET:DESC	CalSet.Description
Change the Contents of a Cal Set		calset object
Recall a Cal Set		app.Recall
Correction Settings		
CORR ON|OFF for a measurement	SENS:CORR	meas.ErrorCORR
Interpolation ON|OFF	SENS:CORR:INT	meas.InterpolateCORR
Extensions ON|OFF	SENS:CORR:EXT	portExtension.State
Port 1 Extensions Value	SENS:CORR:EXT:PORT	portExt.Port1
Port 2 Extensions Value	SENS:CORR:EXT:PORT	portExt.Port2
Receiver A Extensions Value	SENS:CORR:EXT:REC	portExt.InputA
Receiver B Extensions Value	SENS:CORR:EXT:REC	portExt.InputB
Relative Velocity	SENS:CORR:RVEL:COAX	app.VelocityFactor
Modify Cal Kits		
Set a Cal Kit Active	SENS:CORR:COLL:CKIT	app.CalKitType
Get a Handle to the Active Cal Kit		app.ActiveCalKit
Save All Cal Kits after Modifying		app.SaveKits
Load (Recall) All Cal Kits		app.RecallKits
Restore Cal Kit Default	SENS:CORR:COLL:CKIT:RESet	app.RestoreCalKitDefaults
Restore ALL Cal Kits Default		app.RestoreCalKitDefaultsAll
Build a Hybrid Cal Kit		app.BuildHybridKit
Set the Name of a Cal Kit	SENS:CORR:COLL:CKIT:NAME	calKit.Name
Get the Number of Cal Kit		calKit.CalKitType
Set the Port Label of a Cal Kit		calKit.Portlabel
Modify Cal Standards		
Select a Cal Standard	SENS:CORR:COLL:CKIT:STAN	calkit.GetCalStandard
Assign a Class to a Standard	SENS:CORR:COLL:CKIT:ORD1	calKit.StandardForClass
Set Standard Type	SENS:CORR:COLL:CKIT:STAN:TYPE	calstd.Type
Set Delay	SENS:CORR:COLL:CKIT:STAN:DEL	calstd.Delay
Set Loss	SENS:CORR:COLL:CKIT:STAN:LOSS	calstd.loss
Set Impedance	SENS:CORR:COLL:CKIT:STAN:IMP	calstd.Z0
Set Max Frequency	SENS:CORR:COLL:CKIT:STAN:FMAX	calstd.MaximumFrequency
Set Min Frequency	SENS:CORR:COLL:CKIT:STAN:FMIN	calstd.MinimumFrequency
Set Label	SENS:CORR:COLL:CKIT:STAN:LAB	calstd.Label
Set Medium (coax|waveguide)	SENS:CORR:COLL:CKIT:STAN:CHAR	calstd.Medium
Set Capacitance (C0 to C3)	SENS:CORR:COLL:CKIT:STAN:C0	calstd.C0
Set Inductance (L0 to L3)	SENS:CORR:COLL:CKIT:STAN:L0	calstd.L0
Set Arbitrary Impedance (TZReal, TZImag)	SENS:CORR:COLL:CKIT:STAN:TZReal	calstd.TZReal

Power Calibration		
Source Power Cal	Source:Power:CORR	See Power Cal
Receiver Power Cal	Calc:Normalize	See Power Cal
GPIB Power Meter Address	SYST:COMM:GPIB:PMET:ADDR	pwrCal.PowerMeterGPIBAddress
Retrieve and Put Calibration Data		
Retrieve Cal Data from the PNA	CALC:DATA scorr?	see Data Topic
Put Cal Data in the PNA	CALC:DATA scorr	see Data Topic

Marker Commands

Function | Search

Description	SCPI	COM
ON|OFF	CALC:MARK	Marker Object
Delete All Markers	CALC:MARK:AOFF	meas.DeleteAllMarkers
Delete Marker		meas.DeleteMarker
Viewing Marker readouts	View Topic	View Topic
Interpolate All Markers		meas.Interpolate
Interpolate Individ. Marker	CALC:MARK:DISC	mark.Interpolated
Type (Normal | Fixed)	CALC:MARK:TYPE	mark.Type
Format All Markers		meas.MarkerFormat
Format Individ. Marker	CALC:MARK:FORM	mark.Format
Get a handle to Ref marker		meas.GetReferenceMarker
Reference Marker On | Off	CALC:MARK:REF	meas.ReferenceMarkerState
Coupled Markers	CALC:MARK:COUP	app.CoupledMarkers
Delta Marker	CALC:MARK:DELT	mark.DeltaMarker
Read/Set Data Point number		mark.BucketNumber
Read/Set X-axis value	CALC:MARK:X	mark.Stimulus
Read/Set Y-axis value	CALC:MARK:Y	mark.Value
Function		
Marker=> Center, Span, and so forth	CALC:MARK:SET	
Marker=> Center (Freq)		mark.SetCenter
Marker=> CW Freq		mark.SetCW
Marker=> Start (Freq)		mark.SetStart
Marker=> Stop (Freq)		mark.SetStop
Marker=> Elect. Delay		mark.SetElectricalDelay
Marker=> Ref. Level		mark.SetReferenceLevel
Search		
Execute Search	CALC:MARK:FUNC:EXEC	
Select Search Function	CALC:MARK:FUNC	
Maximum	CALC:MARK:FUNC	mark.SearchMax
Minimum	CALC:MARK:FUNC	mark.SearchMin
Target (Value)	CALC:MARK:TARG	mark.TargetValue
Excursion Value	CALC:MARK:FUNC:APE:EXC	mark.PeakExcursion
Threshold Value	CALC:MARK:FUNC:APE:THR	mark.PeakThreshold

Assign Marker to Domain	CALC:MARK:FUNC:DOM:USER	mark.UserRange
Domain Range Start	CALC:MARK:FUNC:DOM:USER:START	mark.UserRangeMin
Domain Range Stop	CALC:MARK:FUNC:DOM:USER:STOP	mark.UserRangeMax
Tracking	CALC:MARK:FUNC:TRAC	mark.Tracking
Bandwidth (Target)	CALC:MARK:TARG	meas.BandwidthTarget
Search Filter Bandwidth	CALC:MARK:BWID	meas.SearchFilterBandwidth
Read Filter BandWidth	CALC:MARK:BWID	meas.FilterBW
Read Filter Center Freq	CALC:MARK:BWID	meas.FilterCF
Read Filter Loss	CALC:MARK:BWID	meas.FilterLoss
Read Filter Q	CALC:MARK:BWID	meas.FilterQ

Trace Commands

Math | Smooth | Stats | Limits | Transform

Description	**SCPI**	**COM**
Create S-Parameter Meas.		app.CreateSParameter
Create Measurement	CALC:PAR:DEF	app.CreateMeasurement
Create Custom Measurement		INACustomMeasurement_Interface
Add Measurement		meass.Add
List Measurements	CALC:PAR:CAT	chans.Count
Delete a Measurement	CALC:PAR:DEL	Measurements.Remove
Get a handle to a Trace		win.ActiveTrace
Select a Measurement	CALC:PAR:SEL	app.ActiveMeasurement
Read Channel Number		chan.ChannelNumber
Read Number of Measurements		chans.Count
Read Measurement Parameter		meas.Parameter
Set / Read Measurement Name		meas.Name
Read Measurement Number		meas.Number
Change Parameter		meas.ChangeParameter
Measurement Format	CALC:FORM	meas.Format
Math		
Data Trace ON|OFF	DISP:WIND:TRAC	
Memory Trace ON|OFF	DISP:WIND:TRAC:MEM	
View Trace Type (Data|Memory|None)		meas.View
Data =>Memory	CALC:MATH:MEM	meas.DataToMemory
Trace Math (Add|Sub|Mult|Div)	CALC:MATH:FUNC	meas.TraceMath
Smoothing		
Smoothing ON|OFF	CALC:SMO	meas.Smoothing
Smoothing Aperture	CALC:SMO:APER	meas.SmoothingAperture

Smoothing Points	CALC:SMO:POIN	
Statistics		
Statistics ON|OFF	CALC:FUNC:STAT	meas.ShowStatistics
Statistics Range	CALC:FUNC:DOM:USER	meas.StatisticsRange
Domain Range Start	CALC:FUNC:DOM:USER:STAR	chan.UserRangeMin
Domain Range Stop	CALC:FUNC:DOM:USER:STOP	chan.UserRangeMax
Set Type (Pk-Pk|StdDev|Mean)	CALC:FUNC:TYPE	
Get All Statistics Data	CALC:FUNC:DATA	meas.GetFileterStatistics
Get Standard Deviation		meas.StandardDeviation
Get Mean		meas.Mean
Get Peak to Peak		meas.PeakToPeak
Limit Lines		
Display Lines ON|OFF	CALC:LIM:DISP:STAT	Limttest.LineDisplay
Fail Sound ON|OFF	CALC:LIM:SOUN	Limttest.SoundOnFail
Testing ON|OFF	CALC:LIM:STAT	Trans.State
Limit Test Failed		meas.LimitTestFailed
Count Limit Lines		chans.Count
Read Test Results	GP-IB_Command_Finder\Status	limts.GetTestResult
Make Limit Lines	CALC:LIM:DATA	
Limit Line Type (Max|Min)	CALC:LIM:SEGM:TYPE	limts.Type
Begin Stimulus	CALC:LIM:SEGM:STIM:STAR	limtseg.BeginStimulus
End Stimulus	CALC:LIM:SEGM:AMPL:STOP	limtseg.EndStimulus
Begin Response	CALC:LIM:SEGM1:AMPL:STAR	limtseg.BeginResponse
End Response	CALC:LIM:SEGM1:AMPL:STOP	limtseg.EndResponse
Transform		
Transform ON|OFF	CALC:TRAN:TIME:STAT	trans.State
Mode (LowPass, BandPass)	CALC:TRAN:TIME	trans.Mode
Start Time	CALC:TRAN:TIME:STAR	trans.Start
Stop Time	CALC:TRAN:TIME:STOP	trans.Stop
Center	CALC:TRAN:TIME:CENT	trans.Center
Span	CALC:TRAN:TIME:SPAN	trans.Span
Step Rise Time	CALC:TRAN:TIME:STAR	trans.StepRiseTime
Set Low Pass Frequency	CALC:TRAN:TIME:LPFR	trans.SetFrequencyLowPass
Gating		
ON|OFF	CALC:FILT:TIME:STAT	gate.State
Type (BandPass, Notch)	CALC:FILT:TIME	gate.Type
Shape	CALC:FILT:GATE:TIME:SHAP	gat.Shape
Start	CALC:FILT:TIME:STAR	gate.Start
Stop	CALC:FILT:GATE:TIME:STOP	gate.Stop
Center	CALC:FILT:GATE:TIME:CENT	gate.Center
Span	CALC:FILT:GATE:TIME:SPAN	gate.Span
Window		

Kaiser Beta	CALC:TRAN:TIME:KBES	trans.KaiserBeta
Impulse Width	CALC:TRAN:TIME:IMP:WIDT	trans.ImpulseWidth

Scale Commands

Description	SCPI	COM
AutoScale	DISP:WIND:TRAC:Y:AUTO	Trce.Autoscale
AutoScale All		Trce.Autoscale
Per Division	DISP:WIND:TRAC:Y:PDIV	trce.YScale
Reference Level	DISP:WIND:TRAC:Y:RLEV	trce.ReferenceValue
Reference Position	DISP:WIND:TRAC:Y:RPOS	trce.ReferencePosition
Electrical Delay	CALC:CORR:EDEL:TIME	meas.ElectricalDelay
Phase Offset	CALC:CORR:OFFS:PHAS	meas.PhaseOffset

System Commands

Status | Events | Macros | Rear Panel

Description	SCPI	COM
Quit application		app.Quit
Preset	SYST:PRES	app.Preset
Reset		app.Reset
Status Commands		
Status Registers	GP-IB\Status	
*OPC;*WAI	GP-IB\Common_Commands	
Events		
AllowAllEvents Method		app.AllowAllEvents
AllowEventCategory Method		app.AllowEventCategory
AllowEventMessage Method		app.AllowEventMessage
AllowEventSeverity Method		app.AllowEventSeverity
DisallowAllEvents Method		app.DisallowAllEvents
MessageText Method		app.MessageText
OnCalEvent		app.OnCalEvent
OnChannelEvent		app.OnChannelEvent
OnDisplayEvent		app.OnDisplayEvent
OnHardwareEvent		app.OnHardwareEvent
OnMeasurementEvent		app.OnMeasurementEvent
OnSCPIEvent		app.OnSCPIEvent
OnSystemEvent		app.OnSystemEvent
OnUserEvent		app.OnUserEvent
SetFailOnOverRange		app.SetFailOnOverRange
Macros		
Execute Macro		app.ExecuteShortcut

Get Macro		app.GetShortcut
Delete Macro		app.DeleteShortCut
Put Macro		app.PutShortcut
Rear Panel Connector Controls		
Material Handler I/O Connector	GP-IB\Control	HWMaterialHandlerIO_Object
Auxiliary IO Connector	GP-IB\Control	HWauxIO_Object
External Test Set Connector	GP-IB\Control	HWExternalTestSetIO_Object
Output Voltage Mode	GP-IB\Control	HWAuxIO2
FootSwitch Mode	GP-IB\Control	HWAuxIO3

Data Commands

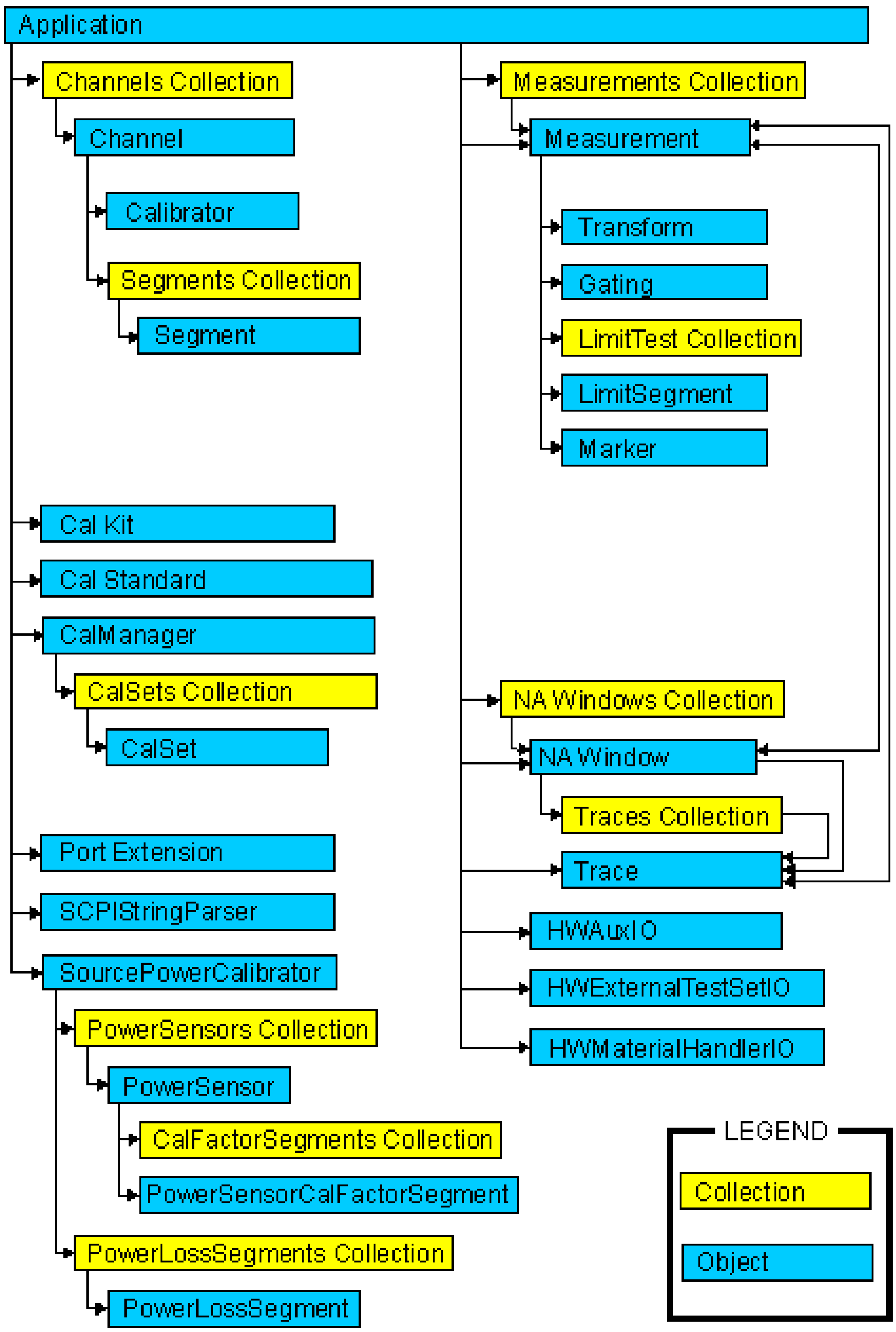
See a map of the data access locations

Description	SCPI	COM
Get Measurement Data FROM the Analyzer		
Get **complex** data from the specified location.		IArrayTrans.getComplex
Get typed **NAComplex** data from the specified location.		IArrayTrans.getNAComplex
Get **data pairs** from the specified location.		IArrayTrans.getPairedData
Get **scalar** data from the specified location.		IArrayTrans.getScalar
Get **variant** data from the specified location		meas.GetData
Specifies ASCII or REAL type for data transfers	Format:Data	
Get complex or formatted data from the measurement or memory result buffer	Calc:Data	
Put Measurement Data INTO the Analyzer		
Put **complex** data into the specified location.		IArrayTrans.putComplex
Put typed **NAComplex** data into the specified location.		IArrayTrans.putNAComplex
Put **scalar** data into the measurement result location.		IArrayTrans.putScalar
Put **complex Variant** data into the specified location.		IArrayTrans.putDataComplex
Put complex or formatted data into the measurement or memory result buffer	Calc:Data	
Get Calibration Data FROM the Analyzer		
Get **complex Error Term** data		ICalData.GetErrorTermComple x
Get **variant Error Term** data	Calc:Data?	CalSet.getErrorTerm
Get **complex Standard** data		ICalData2.getStandardComple

		x
Get **variant Standard** data		CalSet.getStandard
Put Calibration Data INTO the Analyzer		
Put **complex Error Term** data		ICalData.putErrorTermComplex
Put **variant Error Term** data	Calc:Data	CalSet.putErrorTerm
Put **complex Standard** data		ICalData2.putStandardComplex
Put **variant Standard** data		CalSet.putStandard
Get and Put Custom Measurement Data		
Get and Put Custom data	Calc:Data	IArrayTransfer2 Interface

PNA Object Model

See a list of obsolete commands.


Application
Channels Collection
Channel
Calibrator
Segments Collection
Segment
Cal Kit
Cal Standard
CalManager
CalSets Collection
CalSet
Port Extension
SCPIStringParser
SourcePowerCalibrator
PowerSensors Collection
PowerSensor
CalFactorSegments Collection
PowerSensorCalFactorSegment
PowerLossSegments Collection
PowerLossSegment
Measurements Collection
Measurement
Transform
Gating
LimitTest Collection
LimitSegment
Marker
NA Windows Collection
NA Window
Traces Collection
Trace
HWAuxIO
HWExternalTestSetIO
HWMaterialHandlerIO
LEGEND
Collection
Object

Obsolete Commands

As we continue to expand the capability of the PNA, we will continue to develop new COM commands. Some of these new commands replace an existing command, giving it more functionality. Although the existing command will continue to work as usual, we recommend using the new command in code that you develop. Here is a list of replacement commands:

Old Command	**New Command**
Acquire Cal Standard Method	Acquire Cal Standard2 Method
Create SParameter Method	Create SParameterEX Method
Calibrator.getErrorTerm	CalSet.getErrorTerm
Calibrator.getStandard	CalSet.getStandard
Calibrator.putErrorTerm	CalSet.putErrorTerm
Calibrator.putStandard	CalSet.putStandard

Collection Methods and Properties

Common Methods and Properties

The following Methods and Properties are common to Objects and Collections:

Item Method	Returns an object from the collection of objects.
Remove Method	Removes an item from a collection of objects.
Add Method	Adds an object to the collection.
Count Property	Returns the number of items in a collection of objects.
Parent Property	Returns a handle to the parent object of the collection object being referred to in the statement.
State Property	Turns an Object ON and OFF.

Write-only

Item Method

Description Returns an object from the collection of objects.

Note: The order of objects within a collection cannot be assumed.

VB Syntax *Object*[**.Item**]**(***n***)**

Variable **(Type) - Description**

Object Any of the following **(objects):**

CalFactorSegments collection
Cal Sets collection
Channels collection
LimitTest collection
Measurements collection
NaWindows collection
PowerLossSegments collection
PowerSensors collection
Segments collection
Traces collection

.Item	Optional - Item is the default property of a collections object and therefore can be called implicitly. For example, the following two commands are equivalent: Channels.Item(3).Averaging = 1 Channels(3).Averaging = 1
n	**(variant)** - number of the item in the collection. **Note:** the Measurements and Traces collections allow you to specify the name of the measurement as a string. For example: `measCollection("CH_S11_1").InterpolateMarkers`
Return Type	**(Object)**
Default	Not Applicable
Examples	For i = 1 to Traces.Count -1 Traces.Item(i).YScale = .5dB Next i
C++ Syntax	HRESULT Item(VARIANT index, <interface>** pItem)
Interfaces	ICalFactorSegments ICal Sets IChannels ILimitTest IMeasurements INaWindows IPowerLossSegments IPowerSensors ISegments ITraces

Read-only

Count Property

Description	Returns the number of items in a collection of objects.
VB Syntax	*object*.**Count**
Variable	**(Type) - Description**
object	Any of the following **(objects):** Cal Sets collection CalFactorSegments collection Channels collection LimitTest collection Measurements collection NAWindows collection PowerLossSegments collection PowerSensors collection Segments collection Traces collection
Return Type	Long Integer
Default	Not applicable
Examples	numofchans = chans.Count 'return the number of channels -Read
C++ Syntax	HRESULT get_Count(long *p<interface>)
Interface	ICal Sets

ICalFactorSegments
IChannels
ILimitTest
IMeasurements
INAWindows
IPowerLossSegments
IPowerSensors
ISegments
ITraces

Read-only

Parent Property

Description Returns a handle to the parent object of the collection object being refered to in the statement. The parent property allows the user to traverse from an object back up the object hierarchy.

VB Syntax *object*.**Parent**

Variable	**(Type) - Description**
object	Channels collection Channel object Measurements collection NAWindows collection Traces collection Segments collection PowerSensors collection CalFactorSegments collection PowerLossSegments collection

Return Type Object

Default Not Applicable

Examples

parentobj = chans.Parent 'returns a handle to the parent object (Application) of the chans collection. -Read

C++ Syntax

HRESULT get_Parent(IApplication* *pApplication); //IChannels, IChannel, IMeasurements and INAWindows

HRESULT get_Parent(IChannel* *pChannel); //ITraces

HRESULT get_Parent(INAWindow* *pWindow); //ISegments

HRESULT get_Parent(IPowerSensor* *pSensor); //ICalFactorSegments

HRESULT get_Parent(ISourcePowerCalibrator* *pCalibrator); //IPowerSensors and IPowerLossSegments

Interface

IChannels

IChannel

IMeasurements

INAWindows

ITraces

ISegments

IPowerSensors

ICalFactorSegments

IPowerLossSegments

Write-only

Remove Method

Description Removes an item from a collection of objects.

VB Syntax *Object*.**Remove** *item*

Variable **(Type) - Description**

Object Any of the following **(objects)**

CalFactorSegments collection
Cal Sets collection
Measurements collection
NAWindows collection
PowerLossSegments collection
Segments collection

Note: Segments, CalFactorSegments, and PowerLossSegments have an OPTIONAL argument [size] referring to the number of segments to remove, starting with the *item* parameter.

item **(variant)** - Item number to be removed

Return Type Not Applicable

Default Not Applicable

Examples

```
Measurements.Remove 3 'Removes measurement 3
segments.Remove 2,20 'Removes 20 segments (2 - 21)
```

C++ Syntax

```
HRESULT Remove(VARIANT index); //Measurements
HRESULT Remove(VARIANT index); //Cal Sets
HRESULT Remove(long windowNumber); //NAWindows
HRESULT Remove(VARIANT index, long size); //Segments
HRESULT Remove(VARIANT index, long size); //CalFactorSegments
HRESULT Remove(VARIANT index, long size); //PowerLossSegments
```

Interface

IMeasurements

INAWindows

ISegments

ICalFactorSegments

ICal Sets

IPowerLossSegments

Write/Read

State Property

Description	Turns an Object ON and OFF.
VB Syntax	*object*.**State** = *value*
Variable	**(Type) - Description**
object	Applies to any of the following: Gating (object) LimitTest (object) Port Extension (object) Segment (object) Transform (object)
value	**(boolean)** - **0** - Turns *obj* OFF **1** - Turns *obj* ON
Return Type	Long Integer
Default	Depends on the object: **0** - Gating **0** - LimitTest **0** - Port Extension **1** - Segment **0** - Transform
Examples	Seg.State = 1 'Turns the segment object ON -Write tran = Trans.State 'returns the state of Transform -Read
C++ Syntax	HRESULT get_State(VARIANT_BOOL *pVal) HRESULT put_State(VARIANT_BOOL newVal)
Interface	ISegment ITransform IGating ILimitTest IPortExtension

Application Object

Application Object (default interface is IApplication2)

Description

The Application object is the highest object in the analyzer object model. This object presents methods and properties that affect the entire analyzer, rather than a specific channel or measurement. For example, the application object provides the GetIDString method. There's only one ID string for the instrument, unrelated to the channel or parameter being measured. Likewise, the TriggerSignal Property is global to the instrument. You can elect to use an internally generated (free run) trigger or a manual trigger. Either way, that type of trigger generation will be used on all measurements, on all channels. Therefore, it is under the Application object.

This object is unique in that you must Create this object rather than just get a handle to it. See Getting a Handle to an Object.

Note: The **IApplication2 interface** extends the IApplication interface by adding one property, (DisplayAutomationErrors). This property enables the display of automation errors on the screen.

All methods and properties listed below were created on the IApplication interface. IApplication2 is the **default** interface.

Methods	**Description**
ActivateWindow	Makes a window object the Active Window.
AllowAllEvents	Monitors all events
AllowEventCategory	Monitors an event category
AllowEventMessage	Monitors an event
AllowEventSeverity	Monitors an event severity level
BuildHybridKit	Defines the user kit as port1kit + port2kit.
Channel (object)	
CreateCustomMeasurement	Creates a new custom measurement.
CreateMeasurement	Creates a new measurement.
CreateSParameter	OBSOLETE - Use CreateSParameterEx method
CreateSParameterEx	Creates a new S-Parameter measurement with a 3-port load.
DeleteShortCut	Removes a macro (shortcut) from the list of macros
DisallowAllEvents	Monitors NO events
DisplayAutomationErrors	Enables or disables automation error messages from being displayed on the screen. Use the **IApplication2** interface.
DoPrint	Prints the screen to the active Printer.
ExecuteShortcut	Executes a macro (shortcut) stored in the analyzer.
GetAuxIO	Returns a handle to the AuxIO interface
GetCalManager	Returns a handle to the CalManager interface
GetExternalTestSetIO	Returns a handle to the ExternalTestSet IO interface
GetMaterialHandlerIO	Returns a handle to the MaterialHandlerIO interface
GetShortcut	Returns the title and path of the specified macro (shortcut).
LaunchCalWizard	Launches the Cal Wizard
ManualTrigger	Triggers the analyzer when TriggerSignal = naTriggerManual.
MessageText	Returns a message for an eventID
Preset	Resets the analyzer to factory defined default settings.
PrintToFile	Saves the screen data to bitmap (.bmp) file of the screen.
PutShortcut	Puts a Macro (shortcut) file into the analyzer.
Quit	Ends the Network Analyzer application.
Recall	Restores all cal kits from disk.
RecallKits	Recalls the current state of the calibration kits on disk.
Reset	Removes all existing windows and measurements.
RestoreCalKitDefaults	Restores the factory defaults for the specified kit.
RestoreCalKitDefaultsAll	Restores the factory defaults for all kits.
Save	Saves files to disk
SaveKits	Saves all cal kits to disk.
SetFailOnOverRange	Causes over range values to return an error code

ShowStatusBar	Shows and Hides the Status Bar.
ShowStimulus	Shows and Hides Stimulus information.
ShowTitleBars	Shows and Hides the Title Bars.
ShowToolbar	Shows and Hides the specified Toolbar.
Properties	**Description**
ActiveCalKit	Returns a pointer to the kit identified by kitNumber.
ActiveChannel	Returns a handle to the Active Channel object.
ActiveMeasurement	Returns a handle to the Active Measurement object.
ActiveNAWindow	Returns a handle to the Active Window object.
ArrangeWindows	Sets or returns the arrangement of all the windows.
CalKitType	Sets or returns the calibration kit type for to be used for calibration or for kit modification. Shared with the CalKit object.
Channels (collection)	
CoupledMarkers	Sets (or reads) coupled markers ON and OFF
ExternalALC	Sets or returns the source of the analyzer leveling control.
GPIBMode	Makes the analyzer the system controller or a talker/listener.
IDString	Returns the model, serial number and software revision of the analyzer
Measurements (collection)	
NAWindows (collection)	
NumberOfPorts	Returns the number of hardware source ports on the PNA
Options	Returns the options on the analyzer
PortExtension (object)	
SCPIStringParser (object)	
SourcePowerCalibrator (object)	
SourcePowerState	Turns Source Power ON and OFF.
SystemImpedanceZ0	Sets the analyzer impedance value
TriggerDelay	Sets or returns the delay time for a trigger.
TriggerSignal	Sets or returns the trigger source.
TriggerType	Sets or returns the scope of a trigger signal.
VelocityFactor	Sets the velocity factor to be used with Electrical Delay and Port Extensions.
Visible	Makes the Network Analyzer application visible or not visible. **(Default property of this object)**
WindowState	Sets or returns the window setting of Maximized, Minimized, or Normal. Shared with the NAWindow Object
Events	**Description**
OnCalEvent	Triggered by a calibration event.
OnChannelEvent	Triggered by a channel event.
OnDisplayEvent	Triggered by a display event.
OnHardwareEvent	Triggered by a hardware event.
OnMeasurementEvent	Triggered by a measurement event.
OnSCPIEvent	Triggered by a SCPI event.
OnSystemEvent	Triggered by a system event.
OnUserEvent	For future use

Write-only **About Windows**

ActivateWindow Method

Description	Makes a window object the Active Window. In order to change properties on any of the active objects, you must first have a "handle" to the active object using the **Set** command. For more information, See Programming the Analyzer Object Model. You do not have to make an object "Active" to set or read its properties remotely. But an object must be "Active" to change its values from the front panel.
VB Syntax	*app*.**ActivateWindow** *n*
Variable	**(Type) - Description**
app	An Application **(object)**
n	**(long)** Number of the window to make active
Return Type	Window Object
Default	Not Applicable
Examples	app.ActivateWindow 4
C++ Syntax	HRESULT ActivateWindow(long WindowNumber)
Interface	IApplication

Write/Read **About Analyzer Events**

AllowAllEvents Method

Description	Sets event filtering to monitor all events in the analyzer. This is the default setting when subscribing to events. This could slow the measurement speed of the analyzer significantly.
VB Syntax	*app*.**AllowAllEvents**
Variable	**(Type) - Description**
app	An Application **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	*app*.AllowAllEvents
C++ Syntax	HRESULT AllowAllEvents()
Interface	IApplication

Write/Read **About Analyzer Events**

AllowEventCategory Method

Description	Sets event filtering to monitor a category of event.
VB Syntax	*app.***AllowEventCategory,** *category, state*

Variable	**(Type) - Description**
app	An Application **(object)**
category	Category to monitor. Choose from list in Working with the Analyzer's Events
state	**(boolean)** **True** - monitor **False** - do not monitor
Return Type	Not Applicable
Default	Not Applicable

Examples	app.AllowEventCategory

C++ Syntax	HRESULT AllowEventCategory(tagNAEventCategory category, VARIANT_BOOL bAllow)
Interface	IApplication

Write/Read **About Analyzer Events**

AllowEventMessage Method

Description	Sets event filtering to monitor specific events.
VB Syntax	*app.***AllowEventMessage** *event*

Variable	**(Type) - Description**
app	An Application **(object)**
event	Event to monitor. Refer to list in Working with the Analyzer's Events
state	**(boolean)** **True** - monitor **False** - do not monitor
Return Type	Not Applicable
Default	Not Applicable

Examples	app.AllowEventMessage

C++ Syntax	HRESULT AllowEventMessage(tagNAEventID eventID, VARIANT_BOOL bAllow)
Interface	IApplication

Write/Read **About Analyzer Events**

AllowEventSeverity Method

Description	Sets event filtering to monitor levels of severity.
VB Syntax	*app.***AllowEventSeverity** *severity,state*

Variable	**(Type) - Description**
app	An Application **(object)**
severity	**(enum naEventSeverity)** Choose from:naEventSeverityERROR naEventSeverityINFORMATIONAL naEventSeveritySUCCESS naEventSeverityWARNING

state	**(boolean)** **True** - monitor **False** - do not monitor
Return Type	Not Applicable
Default	Not Applicable
Examples	app.AllowEventSeverity
C++ Syntax	HRESULT AllowEventSeverity(tagNAEventSeverity severity, VARIANT_BOOL bAllow)
Interface	IApplication

Write-only **About Modifying Cal Kits**

BuildHybridKit Method

Description	Use this method when you have different port connectors. This is a convenient way to combine two kits that match the connectors on your DUT.
VB Syntax	*app.***BuildHybridKit** *port1Kit,p1sex,port2Kit,p2sex,adapter,user kit*
Variable	**(Type) - Description**
app	An Application **(object)**
port1Kit *port2Kit*	**(enum NACalKit)** - Specifies the two kits to be used to build the hybrid kit. Choose from: naCalKit_85032F_N50 naCalKit_85033E_3_5 naCalKit_85032B_N50 naCalKit_85033D_3_5 naCalKit_85038A_7_16 naCalKit_85052C_3_5_TRL naCalKit_User7 naCalKit_User8 naCalKit_User9 naCalKit_User10
p1sex *p2sex*	**(enum NAPortSex)** - Specifies the sex of the connector at that port. Choose from: **naMale** **naFemale** **naDon'tCare**
adapter	**(enum NAAdapter)** -Choose from: **naUserkit -** the electrical length of the adapter in the userKit specifications **naZeroLength** - no adapter
userKit	**(enum NACalKit)** - The Hybrid kit - Choose from the previous list of kits
Return Type	Not Applicable
Default	Not Applicable
Examples	app.BuildHybridKit naCalKit_85033E_3_5,naMale,naCalKit_85038A_7_16 ,naFemale,naUserkit,naCalKit_User8
C++ Syntax	HRESULT BuildHybridKit\(tagNACalKit port1Kit, tagNAPortSex port1Sex, tagNACalKit port2Kit, tagNAPortSex port2Sex, tagNAAdapter adapter,

	tagNACalKit userKit)
Interface	IApplication

Write-only **About Custom Measurements**

CreateCustomMeasurement Method

Description	Creates a new custom measurement.
VB Syntax	*app.***CreateCustomMeasurement** *chanNum,guid[,window]*
Variable	**(Type) - Description**
app	**(object) -** An Application object
chanNum	**(long)** -Channel number used by the new measurement; can exist or be a new channel.
guid	**(string) -** the GUID (Globally Unique IDentifier) of the new custom measurement object. The new custom measurement must be installed and registered on the PNA. Should be in "registry format". See example below.
window	**(long)** Optional argument. Number of the window the new custom measurement will be placed in. Choose **1** to **4**. If unspecified, the measurement is placed in the active window.
Return Type	Not Applicable
Default	Not Applicable
Examples	app.CreateCustomMeasurement 1, "{12345678-56D3-11D5-AD50-00108334AE98}" 'Not an actual custom measurement - for example purpose only
C++ Syntax	HRESULT CreateCustomMeasurement (long ChannelNum, BSTR guid, long windowNumber)
Interface	IApplication

Write-only **About Measurement Parameters**

CreateMeasurement Method

Description	Creates a new measurement
VB Syntax	*app.***CreateMeasurement** *chanNum,param,lPort[,window]*
Variable	**(Type) - Description**
app	Application **(object)**
chanNum	**(long)** - Channel number of the new measurement; can exist or be a new channel
param	**((string)** - New parameter. Choose from: **S11 I S22 I S21 I S12** Additionally, for 3-port analyzers only: **S33 I S13 I S31 I S23 I S32** For non-ratioed measurements: **A I B I R1 I R2**

C -3-port analyzers only

For ratioed measurements:
A/B

A/C - 3 port analyzers only

B/A

B/C - 3 port analyzers only

C/A - 3 port analyzers only

C/B - 3 port analyzers only

A/R1

B/R1

C/R1 - 3 port analyzers only

A/R2

B/R2

R1/A

R2/A

R1/B

R2/B

R1/C - 3 port analyzers only

R2/R1

R1/R2

lPort	**(long)** - Load port if *param* is a reflection S-Parameter Ignored if *param* is a transmission S-Parameter Source port if *param* is anything other than an S-parameter
window	**(long)** Optional argument. Window number of the new measurement. Choose **1** to **4**. If unspecified, the measurement will be created in the

	Active Window.
Return Type	Not Applicable
Default	Not Applicable
Examples	app.CreateMeasurement(1,"A/R1",1,0)
C++ Syntax	HRESULT CreateMeasurement(long ChannelNum, BSTR strParameter, long lPort, long windowNumber)
Interface	IApplication

Write-only **About Measurement Parameters**

CreateS-Parameter Method - Obsolete

Description	**Note**: This method is replaced by Create SParameterEX method which also allows the selection of a load port This method creates a new S-Parameter measurement in an existing or new window.
VB Syntax	*app.***CreateSParameter** *chan,recvr,source,[window]*
Variable	**(Type) - Description**
app	Application **(object)**
chan	**(long integer)** - Channel number of the new measurement
recvr	**(long integer)** - Port number of the receiver (1 or 2)
source	**(long integer)** - Port number of the source (1 or 2)
window	**(long integer)** - Optional argument. Window number of the new measurement. Choose **1** to **4**. If unspecified, the S-Parameter will be created in the Active Window.
Return Type	Not Applicable
Default	Not Applicable
Examples	app.CreateSParameter 1,2,1,1 'Creates a new S21 measurement in channel 1 and New window(1) app.CreateSParameter 1,2,1 'Creates a new S21 measurement in channel 1 and in the active window
C++ Syntax	HRESULT CreateSParameter(long ChannelNum, long RcvPort, long SrcPort, long windowNumber)
Interface	IApplication

Write-only **About Measurement Parameters**

CreateSParameterEx Method

Description	Creates a new S-Parameter measurement in an existing or new window and specifies the load port for 3-port devices.
VB Syntax	*app.***CreateSParameter** *chan,recvr,source[,loadPort][,window]*
Variable	**(Type) - Description**
app	Application **(object)**
chan	**(long integer)** - Channel number of the new measurement
recvr	**(long integer)** - Port number of the receiver
source	**(long integer)** - Port number of the source
loadPort	**(long integer)** - Port number of the load. Required for reflection

	measurements of 3-port devices on 3-port PNAs.
window	**(long integer)** - Optional argument. Window number of the new measurement. Choose **1** to **4**. If unspecified, the S-Parameter will be created in the Active Window.
Return Type	Not Applicable
Default	Not Applicable
Examples	app.CreateSParameter 1,2,1,1 'Creates a new S21 measurement in channel 1 and New window(1) app.CreateSParameter 2,1,1,3,1 'Creates a new S11 measurement on channel 2 with port 3 as the load. Create in the active window
C++ Syntax	HRESULT CreateSParameter(long ChannelNum, long RcvPort, long SrcPort, long LoadPort, long windowNumber)
Interface	IApplication

Write-only **About Macros**

DeleteShortCut Method

Description	Removes a macro from the list of macros in the analyzer. Does not remove the file. **Note:** There are always 12 macro positions. They do not have to be sequential. For example, you can have number 7 but no numbers 1 to 6.
VB Syntax	*app*.**DeleteShortCut** *item*
Variable	**(Type) - Description**
app	An Application **(object)**
item	**(long integer)** number of the macro to be deleted.
Return Type	Not Applicable
Default	Not Applicable
Examples	*app*.DeleteShortCut 2
C++ Syntax	HRESULT DeleteShortcut(long Number)
Interface	IApplication

Write/Read **About Analyzer Events**

DisallowAllEvents Method

Description	Sets event filtering to monitor NO eventst.
VB Syntax	*app*.**DisallowAllEvents**
Variable	**(Type) - Description**
app	An Application **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	*app*.DisallowAllEvents
C++ Syntax	HRESULT DisallowAllEvents()

Interface	IApplication

Write-Read

DisplayAutomationErrors Property

Description	Enables or disables automation error messages from being displayed on the screen.
VB Syntax	app.**DisplayAutomationErrors** = value
Variable	**(Type) - Description**
app	An application**(object)**
value	(Boolean) **True** allows error to show on display, **False** turns error off from display.
Return Type	Boolean
Default	True
Examples	**Dim app As Application** **Set app = New Application** **app.DisplayAutomationErrors = False 'Turns off display** print app.DisplayAutomationErrors 'prints False
C++ Syntax	HRESULT get_DisplayAutomationErrors(VARIANT_BOOL * Val); HRESULT put_DisplayAutomationErrors(VARIANT_BOOL Val);
Interface	IApplication2

Write-only **About Printing**

DoPrint Method

Description	Prints the screen to the default Printer.
VB Syntax	*app.***DoPrint**
Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	app.DoPrint
C++ Syntax	HRESULT DoPrint()
Interface	IApplication

Write-only **About Macros**

ExecuteShortcut Method

Description	Executes a Macro (shortcut) stored in the analyzer. Use app.getShortcut to list existing macros. Use app.putShortcut to associate the macro number with the file.

VB Syntax	*app*.**ExecuteShortcut** *index*

Variable	**(Type) - Description**
app	Application **(object)**
index	**(long integer)** - Number of the macro stored in the analyzer.
Return Type	Not Applicable
Default	Not Applicable

Examples	app.ExecuteShortcut 1

C++ Syntax	HRESULT ExecuteShortcut(long index)
Interface	IApplication

Read-only **About the AuxIO connector**

GetAuxIO Method

Description	This method returns the IAuxIO interface.
VB Syntax	*app*.**GetAuxIO**

Variable	**(Type) - Description**
app	**(object) -** An Application object
Return Type	IHWAuxIO
Default	Not Applicable

Example	Dim app As AgilentPNA835x.Application Dim aux As HWAuxIO Set aux = app.GetAuxIO

C++ Syntax	HRESULT GetAuxIO (IHWAuxIO **pAux);
Interface	IApplication

Read-only **About Cal Sets**

GetCalManager Method

Description	This method returns the ICalManager interface.
VB Syntax	*app*.**GetCalManager**

Variable	**(Type) - Description**
app	Application **(object)**
Return Type	ICalManager*
Default	Not Applicable

Example	dim app as AgilentPNA835x.Application dim mgr as CalManager set mgr = app.GetCalManager

C++ Syntax	HRESULT GetCalManager(ICalManager **mgr);
Interface	IApplication

Read-only **About the External TestSet connector**

Get ExternalTestSetIO Method

Description	This method returns the IExternalTestSetIO interface.
VB Syntax	*app.*GetExternalTestSetIO
Variable	**(Type) - Description**
app	Application **(object)**
Return Type	IHWExternalTestSetIO
Default	Not Applicable
Example	Dim app As AgilentPNA835x.Application Dim ets As HWExternalTestSetIO Set ets = app.GetExternalTestSetIO
C++ Syntax	HRESULT GetExternalTestSetIO (IHWExternalTestSetIO **ptestset);
Interface	IApplication

Read-only **About the MaterialHandler connector**

Get MaterialHandlerIO Method

Description	This method returns the MaterialHandlerIO interface.
VB Syntax	*app.*GetMaterialHandlerIO
Variable	**(Type) - Description**
app	Application **(object)**
Return Type	IHWMaterialHandlerIO
Default	Not Applicable
Example	Dim app As AgilentPNA835x.Application Dim hand As HWMaterialHandlerIO Set hand = app.GetMaterialHandlerIO
C++ Syntax	HRESULT GetMaterialHandlerIO (IHWMaterialHandlerIO **phand);
Interface	IApplication

Read-only

GetShortcut Method

Description	Returns the Title, Path, and optional argument strings, of the specified Macro (shortcut). Use this method to list the titles and paths of macros in the analyzer.
VB Syntax	*app.*GetShortcut *index, title, path, arguments*
Variable	**(Type) - Description**
app	Application **(object)**

index	**(long)** - Number of the macro. Use a number between **1** and **12.**
title	**(string)** - **Title** of the specified macro. (Appears in the softkey label)
path	**(string)** - **Pathname** of the specified macro.
arguments	**(string)** - Arguments for the specified macro
Return Type	String
Default	Not Applicable

Example

```
Dim t As String
 Dim p As String
 Dim arg As String
 Dim i As Integer
 For i = 1 to 12
  app.GetShortcut i,t,p,arg
  Print t,p
 Next
```

C++ Syntax HRESULT GetShortcut(long Number, BSTR* title, BSTR* pathname, BSTR* arguments)

Interface IApplication

Remarks Shortcuts can also be defined and accessed using the macro key on the front panel. However, the benefit of this feature is primarily for the interactive user

Read-Write **About the Cal Wizard**

LaunchCalWizard Method

Description Launches the Cal Wizard on the PNA and does not return until the Cal Wizard is dismissed.

Note: The Cal Wizard operates on the active measurement. Therefore, activate the measurement to be calibrated before launching the Cal Wizard.

VB Syntax *success = app.***LaunchCalWizard** *(newCS)*

Variable	**(Type) - Description**
success	**(boolean) -** variable to store the returned value **True -** The Cal was completed **False -** The Cal was canceled without completing the calibration.
app	**(object)** Application object
newCS	**(boolean)** **True** - Cal will be performed on a new Cal Set. **False** - Cal will be performed using the existing Cal Set assigned to the channel. If no Cal Set is found, a new Cal Set will be created.
Return Type	Boolean
Default	Not Applicable

Example

```
dim bSuccess as boolean
 dim bNewCalset as boolean
 bNewCalSet = false
 bSuccess = app.LaunchCalWizard( bNewCalSet)
```

C++ Syntax **HRESULT**

Interface IApplication

Write-only **About Triggering**

ManualTrigger Method

Description	Triggers the analyzer when `TriggerSignal = naTriggerManual`.
VB Syntax	*app*.**ManualTrigger** *[sync],[timeout]*
Variable	**(Type) - Description**
app	Application **(object)**
sync	**(boolean)** - Optional argument. A variable set to either True or False. **True** - The analyzer waits until the trigger is completed to process subsequent commands. **False** - Subsequent commands are processed immediately (the default setting)
timeout	**(long) -** Optional argument. If *sync* is true, *timeout* sets the amount of time the PNA will wait until continuing program execution. Units are milliseconds. A value of -1 (the default setting) causes the PNA to wait indefinitely. If sync is False, the timeout setting is ignored.
Return Type	Not Applicable
Default	Not Applicable
Examples	' After Manual trigger is executed, the PNA will wait 1 second to continue program execution Dim wait as Boolean wait = True app.ManualTrigger wait, 1000
C++ Syntax	HRESULT ManualTrigger(VARIANT_BOOL bSynchronize, long timeout)
Interface	IApplication

Write/Read **About Analyzer Events**

MessageText Method

Description	Returns text for the specified eventID
VB Syntax	*app*.**MessageText,***eventID,message*
Variable	**(Type) - Description**
app	An Application **(object)**
enentID	**(enum naEventID)** Choose from the list in Working with the Analyzer's Events
message	**(string) -** variable to store the returned message
Return Type	String
Default	Not Applicable
Examples	RFNA.MessageText naEventID_ARRANGE_WINDOW_EXCEED_CAPACITY, message
C++ Syntax	HRESULT MessageText(tagNAEventID msgID, BSTR* message)
Interface	IApplication

Write-only | **Factory Preset Settings**

Preset Method

Description	**Application Object**: Deletes all traces and windows. In addition, resets the analyzer to factory defined default settings and creates an S11 measurement named "CH1_S11_1" in window 1. **Channel Object:** Resets the channel (object) to factory defined default settings. Does NOT delete the current measurements or add a new measurement.
VB Syntax	*app.***Preset** *chan***.Preset**
Variable	**(Type) - Description**
app	An Application **(object)**
chan	A Channel **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	app.Preset
C++ Syntax	HRESULT Preset()
Interface	IApplication IChannel

Write-only | **About Saving to File**

PrintToFile Method

Description	Saves the screen image to a bitmap (.bmp) file.
VB Syntax	*app.***PrintToFile** *filename*
Variable	**(Type) - Description**
app	An Application **(object)**
filename	**(string)** Name of the file to save the screen to. The file is saved to the current working directory unless a valid full path name is specified. Use one of the following suffixes: • .bmp - not recommended due to large file size • .jpg - not recommended due to poor quality .png - recommended
Return Type	Not Applicable
Default	Not Applicable
Examples	app.PrintToFile "myfile.png" app.PrintToFile "c:\data\myfile.png"

C++ Syntax HRESULT PrintToFile(BSTR bstrFile)
Interface IApplication

Write-only **About Macros**

PutShortcut Method

Description Defines a Macro (shortcut) file in the analyzer. This command links a file name and path to the Macro file. You still need to put the macro file in the analyzer at the location indicated by this command.

VB Syntax *app.***PutShortcut** *index,title,path*

Variable	**(Type) - Description**
app	Application **(object)**
index	**(long)** - Number of the macro to be stored in the analyzer. If the index number already exists, the existing macro is replaced with the new macro.
title	**(string)** - The name to be assigned to the macro
path	**(string)** - Full pathname to the existing macro "executable" file.
Return Type	Not Applicable
Default	Not Applicable

Examples app.PutShortcut 1,"Test","C:\Automation\MyTest.vbs"

C++ Syntax HRESULT PutShortcut(long Number, BSTR title, BSTR pathname)
Interface IApplication

Write-only

Quit Method

Description Terminates the Network Analyzer application.

VB Syntax *app.***Quit**

Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples app.Quit

C++ Syntax HRESULT Quit()
Interface IApplication
Remarks

Under the rules of COM, the server should not exit until all references to it have been released. This method is a brute force way of terminating the application. Be sure to release all references (or terminate the client program) before attempting to restart the Network Analyzer application.

An alternate approach to terminating the application is to make the application invisible (app.Visible = False) and release all references. The server will shutdown.

Write-only **About Save/Recall**

Recall Method

Description	Recalls a measurement state, calibration state, or both from the hard drive into the analyzer. Use app.Save to save the measurement and calibration state.
VB Syntax	*app*.**Recall** (*filename.ext*)
Variable	**(Type) - Description**
app	Application **(object)**
filename.ext	**(string)** - Filename and extension of the file to be recalled. Extensions: • .sta - Instrument State • .cal - Calibration file • .cst - Both Instrument State and Calibration file Files are stored in the default folder "C:\Program Files\Agilent\Network Analyzer\Documents To recall from a different folder, specify the pathname in the *filename.ext.* argument.
Return Type	Not Applicable
Default	Not Applicable
Examples	app.Recall (mystate.sta) 'Recalls "mystate.sta" from the default folder app.Recall ("C:\Program Files\Agilent\Network Analyzer\Documents\Newfolder\MyState.cst) 'Recalls "mystate.cst" from the specified folder
C++ Syntax	HRESULT Recall(BSTR bstrFile)
Interface	IApplication

Write-only **About Modifying Cal Kits**

Recall Kits Method

Description	Recalls the calibration kits definitions that were stored witht the SaveKits command.
VB Syntax	*app*.**RecallKits**
Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	app.RecallKits
C++ Syntax	HRESULT RecallKits()
Interface	IApplication

Write-only **About Presetting the Analyzer**

Reset Method

Description	Removes all existing windows and measurements from the application. (Unlike Preset, does not create a new measurement.)
VB Syntax	*app.***Reset**

Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples	app.Reset

C++ Syntax	HRESULT Reset()
Interface	IApplication

Write-only **About Modifying Cal Kits**

RestoreCalKitDefaults Method

Description	Restores the original properties of the specified Cal Kit, overwritting the last definition with the factory defaults.
VB Syntax	*app.***RestoreCalKitDefaults** (*calKit*)

Variable	**(Type) - Description**
app	Application **(object)**
calKit	**(enum NACalKit)** - Calibration Kit to restore. Choose from: 1 - naCalKit_85032F_N50 2 - naCalKit_85033E_3_5 3 - naCalKit_85032B_N50 4 - naCalKit_85033D_3_5 5 - naCalKit_85038A_7_16 6 - naCalKit_85052C_3_5_TRL 7 - naCalKit_User7 8 - naCalKit_User8 9 - naCalKit_User9 10 - naCalKit_User10
Return Type	Not Applicable
Default	Not Applicable

Examples	app.RestoreCalKitDefaults naCalKit_MechKit10

C++ Syntax	HRESULT RestoreCalKitDefaults\(tagNACalKit kit)
Interface	IApplication

Write-only **About Modifying Cal Kits**

RestoreCalKitDefaultsAll Method

Description	Restores the original properties of ALL of the Cal Kits, overwritting the last definitions with the factory defaults.
VB Syntax	*app.***RestoreCalKitDefaultsAll**

Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples app.RestoreCalKitDefaultsAll

C++ Syntax HRESULT RestoreCalKitDefaultsAll()
Interface IApplication

Write-only **About Save/Recall**

Save Method

Description Saves the appropriate content to the hard drive depending on the file suffix provided. See the table below. Some saved files can be recalled using app.Recall. depending on the content.

VB Syntax *app.*__Save__(*filename.ext*)

Variable **(Type) - Description**

app Application **(object)**

filename.ext **(string)** - Filename and extension of the file to be saved.

Extensions:

- **.cst** - Saves both Instrument State and Cal Set reference - Recalls a calibrated measurement. (Recallable)
- **.sta** - Saves Instrument State only - recalls the instrument state without calibration. (Recallable)
- **.cal** - Calibration file – saves the active Cal Sets currently in use by any channel. Use this mode for archival purposes only. All Cal Sets are saved to a Cal Set data file. This mode provides a method of safeguarding calibration data. This data can be restored to the list of Cal Sets available in the instrument. (Recallable)
- **.prn** - Saves active trace in comma-separated format (not recallable)
- **.bmp** - Saves a Bitmap of the screen (not recallable)
- **.s1p** - Saves 1-port measurement data (not recallable)
- **.s2p** - Saves 2-port measurement data (not recallable)

Files are saved to the default folder "C:\Program Files\Agilent\Network Analyzer\Documents. To save to a different folder, specify the pathname in the *filename.ext*. argument.

Return Type Not Applicable
Default Not Applicable

Examples app.Save(mystate.sta) 'Saves "mystate.sta" to the default folder
app.Save("C:\Program Files\Agilent\Network Analyzer\Documents\Newfolder\MyState.cst) 'Saves "mystate.cst" to the specified folder

C++ Syntax HRESULT Save(BSTR bstrFile)
Interface IApplication

Write-only **About Modifying Cal Kits**

SaveKits Method

Description	Saves the cal kits, typically after modifying a calibration kit. To load a cal kit into the analyzer from the hard drive, use app.RecallKits.
VB Syntax	*app.***SaveKits**
Variable	**(Type) - Description**
app	Application **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	app.SaveKits
C++ Syntax	HRESULT SaveKits()
Interface	IApplication

Write/Read **About Analyzer Events**

SetFailOnOverRange Method

Description	When set TRUE, configures the analyzer to report outOfRange conditions with an **error** code. Any overrange error will return **E_NA_LIMIT_OUTOFRANGE_ERROR**. **Note:** This method is for the benefit of VB clients. The analyzer automatically adjusts overrange conditions to the closest acceptable setting. The VB user will not See that an overrange occurred because the HRESULT is not returned if it has a success code. For more information, See Events/OverRange.
VB Syntax	*app.***SetFailOnOverRange** *state*
Variable	**(Type) - Description**
app	An Application **(object)**
state	**(boolean)** - **True (1)** - Overrange conditions report an error code **False (0)** - Overrange conditions report a success code
Return Type	Not Applicable
Default	False (0)

VB Example

```
app.SetFailOnOverRange TRUE
 On Error Goto ERRHANDLER

 'the following overrange will cause ERRHANDLER to be
invoked

 channel.StartFrequency = 9.9 GHZ
 exit
ERRHANDLER:
  print "something failed"
```

C++ Syntax HRESULT put_SetFailOnOverRange(VARIANT_BOOL mode)
Interface IApplication

Write-only **About Display Formatting**

ShowStatusBar Method

Description	Shows and Hides the Status Bar. The Status Bar is located across the bottom of the display. The following information is shown for the active measurement: • Channel number • Parameter • Correction On or Off Remote or Local operation
VB Syntax	*app*.**ShowStatusBar** *state*
Variable	**(Type) - Description**
app	Application **(object)**
state	**(boolean)** - **True (1)** - Show the Status Bar **False (0)** - Hide the Status Bar
Return Type	Not Applicable
Default	Not Applicable
Examples	app.ShowStatusBar True
C++ Syntax	HRESULT ShowStatusBar (VARIANT_BOOL bState)
Interface	IApplication

Write-only **About Display Formatting**

ShowStimulus Method

Description	Shows and Hides the Stimulus (X-axis) information located at the bottom of the display. The start and stop stimulus values are shown for the active measurement.
VB Syntax	*app*.**ShowStimulus** *state*
Variable	**(Type) - Description**
app	Application **(object)**
state	**(boolean)** - **True (1)** - Show the Stimulus information **False (0)** - Hide the Stimulus information
Return Type	Not Applicable
Default	Not Applicable
Examples	app.ShowStimulus True
C++ Syntax	HRESULT ShowStimulus(VARIANT_BOOL bState)
Interface	IApplication

Write-only **About Display Formatting**

ShowTitleBars Method

Description	Shows and Hides the Title Bars. The Title Bars are across the top of the Network Analyzer Window and each of the measurement windows. The Window name is shown in the Title Bar.
VB Syntax	*app*.**ShowTitleBars** *state*
Variable	**(Type) - Description**
app	Application **(object)**
state	**(boolean)** **True (1)** - Show the Title Bars **False (0)** - Hide the Title Bars
Return Type	Not Applicable
Default	Not Applicable
Examples	app.ShowTitleBars True
C++ Syntax	HRESULT ShowTitleBars\(VARIANT_BOOL bState)
Interface	IApplication

Write-only **About Display Formatting**

ShowToolbar Method

Description	Shows and Hides the specified Toolbar.
VB Syntax	*app*.**ShowToolbar** *toolbar,state*
Variable	**(Type) - Description**
app	Application **(object)**
toolbar	**(enum NAToolbarType)** - The toolbar to show or hide. Choose from: 0 - naToolbar_None 1 - naToolbar_ActiveEntry 2 - naToolbar_Markers 3 - naToolbar_Measurement 4 - naToolbar_Stimulus 5 - naToolbar_SweepControl
state	**(boolean)** - **True (1)** - Show the specified toolbar **False (0)** - Hide the specified toolbar
Return Type	Not Applicable
Default	1 - naToolbar_ActiveEntry showing; all others hiding.
Examples	app.ShowToolbar 1,1 'shows the active entry toolbar
C++ Syntax	HRESULT ShowToolbar(tagNAToolbarType toolbar, VARIANT_BOOL bState)
Interface	IApplication

Read-only **About Calibration Kits**

ActiveCalKit Property

Description	Returns a handle to the Active CalKit object. You can either **(1)** use the handle directly to access CalKit properties and methods, or **(2)** set a variable to the CalKit object. The variable retains a handle to the original object if another CalKit becomes active.
VB Syntax	1) app.**ActiveCalKit.***<setting>* **or** 2) Set *cKit = app.***ActiveCalKit**

Variable	**(Type) - Description**
app	**(object)** - An Application object
<setting>	A CalKit property (or method) and arguments
cKit	**(object)** - A CalKit object
Return Type	CalKit object
Default	None

Examples	Public cKit as calKit Set cKit = app.ActiveCalKit 'read
C++ Syntax	HRESULT get_ActiveCalKit (ICalkit * kit)
Interface	IApplication

Read-only **About Channels**

ActiveChannel Property

Description	Returns a handle to the Active Channel object. You can either **(1)** use the handle directly to access channel properties and methods, or **(2)** set a variable to the channel object. The variable retains a handle to the original channel if another channel becomes active.
VB Syntax	(1) *app.***ActiveChannel.***<setting>* **or** (2) Set *chan = app.***ActiveChannel**

Variable	**(Type) - Description**
chan	A Channel **(object)**
app	An Application **(object)**
<setting>	A channel property (or method) and arguments
Return Type	Channel object
Default	Not applicable

Examples	1) app.ActiveChannel.Averaging = 1 2) Public chan as Channel Set chan = app.ActiveChannel
C++ Syntax	HRESULT get_ActiveChannel(IChannel* *pVal)
Interface	IApplication

Read-only

ActiveMeasurement Property

Description	Returns a handle to the Active Measurement object. You can either **(1)** use the handle directly to access measurement properties and methods, or **(2)** set a variable to the measurement object. The variable retains a handle to the original measurement.
VB Syntax	1) *app.***ActiveMeasurement.***<setting>* **or** 2) Set *meas = app.***ActiveMeasurement**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
app	An Application **(object)**
<setting>	A measurement property (or method) and arguments
Return Type	Measurement object
Default	None
Examples	1) app.ActiveMeasurement.Averaging = 1 2) Public meas as Measurement Set meas = app.ActiveMeasurement
C++ Syntax	HRESULT get_ActiveMeasurement(IMeasurement **ppMeas)
Interface	IApplication

Read-only **About Windows**

ActiveNAWindow Property

Description	Returns a handle to the Active Window object. You can either **(1)** use the handle directly to access window properties and methods, or **(2)** set a variable to the window object. The variable retains a handle to the original window if another window becomes active.
VB Syntax	1) *app.***ActiveNAWindow***.<setting>* **or** 2) Set *win = app.***ActiveNAWindow**
Variable	**(Type) - Description**
win	A NAWindow **(object)**
app	An Application **(object)**
<setting>	A NAWindow property (or method) and arguments
Return Type	A NAWindow object
Default	Not applicable
Examples	Public win as NAWindow Set win = app.ActiveWindow
C++ Syntax	HRESULT get_ActiveNAWindow(INAWindow **ppWindow)
Interface	IApplication

Write/Read **About Arrange Windows**

ArrangeWindows Property

Description	Sets or returns the arrangement of all the windows. Overlay, Stack2, Split3 and Quad4 will create windows. To control the state of the one window you have a handle to, use app.WindowState.
VB Syntax	*app.***ArrangeWindows** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(enum NAWindowModes)** - Choose from: **0 - naTile** **1 - naCascade** **2 - naOverlay** **3 - naStack2** **4 - naSplit3** **5 - naQuad4**
Return Type	NAWindowModes
Default	naTile
Examples	app.ArrangeWindow = naTile 'Write arrWin = app.ArrangeWindows 'Read
C++ Syntax	HRESULT put_ArrangeWindows(tagNAWindowModes newVal)
Interface	IApplication

Write/Read **About Modifying Cal Kits**

CalKitType Property

Description	Sets and returns a calibration kit type for calibration or to be used for kit modification. To get a handle to this kit, use app.ActiveCalKit
VB Syntax	*object*.**CalKitType** = *value*
Variable	**(Type) - Description**
object	A calkit **(object) or** An Application **(object).** **Note: app**.CalKitType and **calkit**.calKitType perform exactly the same function.
value	**(enum naCalKit)** - Calibration Kit type. Choose from: 1 - naCalKit_User1 2 - naCalKit_User2 4 - naCalKit_User3 5 - naCalKit_User4 .. .. .. 49 - naCalKit_User49 50 - naCalKit_User50
Return Type	NACalKit

Default Not Applicable

Examples

calkit.CalKitType = naCalKit_User27
kitype = app.CalKitType

C++ Syntax

HRESULT get_CalKitType(tagNACalKit *pVal)
HRESULT put_CalKitType(tagNACalKit newVal)

Interface

IApplication
ICalKit

Write/Read **About Coupled Markers**

CoupledMarkers Property

Description Sets and Reads the state of Coupled Markers (ON and OFF)

VB Syntax *app*.**CoupledMarkers** = *state*

Variable	**(Type) - Description**
app	**(object) -** An Application object
state	**(boolean)** **False (0)** - Turns Coupled Markers OFF **True (1)** - Turns Coupled Markers ON
Return Type	Boolean **0** - OFF **1** - ON
Default	OFF (0)

Examples

app.CoupledMarkers = True 'Write
coupl = app.CoupledMarkers 'Read

C++ Syntax

Interface IApplication

Write/Read

ExternalALC Property

Description Sets or returns the source of the analyzer leveling control.

VB Syntax *app*.**ExternalALC** = *value*

Variable	**(Type) - Description**
app	An Application **(object)**
value	**(boolean)** - Choose from: **True (or 1)** - Leveling control supplied through the rear panel. **False (or 0)** - Leveling control supplied inside the analyzer
Return Type	Boolean
Default	0

Examples

app.ExternalALC = True 'Write
extALC = app.ExternalALC 'Read

C++ Syntax	HRESULT get_ExternalALC(VARIANT_BOOL *pVal) HRESULT put_ExternalALC(VARIANT_BOOL newVal)
Interface	IApplication

Write/Read **About GPIB Fundamentals**

GPIBMode Property

Description	Changes the analyzer to a GPIB system controller or a talker/listener on the bus. The analyzer must be the controller if you want to use it to send commands to other instruments. The analyzer must be a talker/listener if you want to send it commands from another PC.
VB Syntax	*app.***GPIBMode** *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(enum NAGPIBMode)** -Choose either: **0 -** naTalkerListener - the analyzer is a talker / listener **1 -** naSystemController - the analyzer is the system controller
Return Type	Long Integer
Default	0 - naTalkerListener
Examples	app.GPIBMode = naTalkerListener 'Write mode = app.GPIBMode 'Read
C++ Syntax	HRESULT get_GPIBMode(tagGPIBModeEnum* eGpibMode) HRESULT put_GPIBMode(tagGPIBModeEnum eGpibMode)
Interface	IApplication

Read-only

IDString Property

Description	Returns the ID of the analyzer, including the Model number, Serial Number, and the Software revision number.
VB Syntax	*value = app.***IDString**
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(string)** - variable to contain the returned ID string
Return Type	String
Default	Not Applicable
Examples	id = app.IDString
C++ Syntax	HRESULT IDString(BSTR* IDString)
Interface	IApplication

Read-only

NumberOfPorts Property

Description	Returns the number of hardware source ports on the PNA.
VB Syntax	*value = app.***NumberOfPorts**
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(long integer)** - variable to contain the returned value
Return Type	(long integer)
Default	Not Applicable
Examples	iNumPorts = app.NumberOfPorts
C++ Syntax	HRESULT NumberOfPorts(long* NumPorts)
Interface	IApplication

Read-only **About Options**

Options Property

Description	Returns a string identifying the analyzer option configuration.
VB Syntax	*value = app.***Options**
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(string)** - variable to contain the returned string
Return Type	String
Default	Not Applicable
Examples	availOptions = app.Options
C++ Syntax	HRESULT Options(BSTR* OptionString)
Interface	IApplication

Write/Read **About Source Power**

SourcePowerState Property

Description	Turns Source Power ON and OFF
VB Syntax	*app.***SourcePowerState** = *state*
Variable	**(Type) - Description**
app	An Application **(object)**
state	**(boolean)** **False (0)** - Turns Source Power OFF **True (1)** - Turns Source Power ON

Return Type	Boolean **0** - Power OFF **1** - Power ON
Default	ON (1)
Examples	app.SourcePowerState = True 'Write pwr = app.SourcePowerState 'Read
C++ Syntax	HRESULT get_SourcePowerState(VARIANT_BOOL *pVal) HRESULT put_SourcePowerState(VARIANT_BOOL newVal)
Interface	IApplication

Write/Read **About System Impedance**

SystemImpedanceZ0 Property

Description	Sets and returns the impedance for the analyzer.
VB Syntax	*app*.**SystemImpedanceZ0** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	(double) Analyzer Impedance. Choose any number between 0 and 1000 ohms.
Return Type	Double
Default	50
Examples	app.SystemImpedanceZ0 = 75 'Write z0 = app.SystemImpedanceZ0 'Read
C++ Syntax	HRESULT get_SystemImpedanceZ0(double dSystemZ0) HRESULT put_SystemImpedanceZ0(double *pdSystemZ0)
Interface	IApplication

Write/Read **About Trigger**

TriggerDelay Property

Description	Sets and reads the trigger delay. This delay is only applied while in External Trigger mode. After an external trigger is applied, the start of the sweep is delayed for the specified delay value plus any inherent latency.
VB Syntax	*app*.**TriggerDelay** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**Double**- Trigger delay value. Range is from 0 to 1 second
Return Type	Double
Default	0
Examples	app.TriggerDelay = .003 'Write

delay = app.TriggerDelay 'Read

C++ Syntax HRESULT get_TriggerDelay(delay);
HRESULT put_TriggerDelay(.003)

Interface IApplication

Write/Read **About Trigger Source**

TriggerSignal Property

Description Sets or returns the trigger source.

VB Syntax *app*.**TriggerSignal** = *value*

Variable **(Type) - Description**

app An Application **(object)**

value **(enum NATriggerSignal)** - Choose from:

0 - naTriggerInternal - free run

1 - naTriggerExternalPositive - a trigger signal is generated when a TTL high is sensed on the external trigger pin of the Aux IO connector

2 - naTriggerExternalNegative - a trigger signal is generated when a TTL low is sensed on the external trigger pin of the Aux IO connector.

3 - naTriggerManual - manual trigger source; use `app.ManualTrigger` to send a trigger signal.

4 - **naTriggerExternalHigh -** a trigger signal is generated when a TTL high is sensed on the external trigger pin of the Aux IO connector

5 - **naTriggerExternalLow -** a trigger signal is generated when a TTL low is sensed on the external trigger pin of the Aux IO connector

Return Type Long Integer

Default naTriggerInternal

Examples

app.TriggerSignal = naTriggerExternalPositive 'Write
trigsign = app.TriggerSignal 'Read

C++ Syntax HRESULT get_TriggerSignal(tagNATriggerSignal *pSignal)
HRESULT put_TriggerSignal(tagNATriggerSignal signal)

Interface IApplication

Write/Read. **About Trigger**

TriggerType Property

Description Sets or returns the trigger type which determines the scope of a trigger signal.

Note: naGlobalTrigger is not compatible with TriggerMode = naTriggerModePoint. If you set TriggerType = naGlobalTrigger, any

	channel in naTriggerModePoint will be set to naTriggerModeMeasurement .
VB Syntax	*app*.**TriggerType** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(enum NATriggerType)** - Trigger type. Choose from: **0 - naGlobalTrigger** - a trigger signal is applied to all triggerable channels **1 - naChannelTrigger -** a trigger signal is applied to the current channel. The next trigger signal will be applied to the next channel; not necessarily channel 1-2-3-4.
Return Type	Long Integer
Default	naGlobalTrigger
Examples	app.TriggerType = naGlobalTrigger 'Write trigtyp = app.TriggerType 'Read
C++ Syntax	HRESULT get_TriggerType(tagNATriggerType *pTrigger) HRESULT put_TriggerType(tagNATriggerType trigger)
Interface	IApplication

Write/Read **About Port Extensions**

VelocityFactor Property

Description	Sets the velocity factor to be used with Electrical Delay and Port Extensions.
VB Syntax	*app*.**VelocityFactor** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(double)** - Velocity factor. Choose a number between: **0** and **10** (.66 polyethylene dielectric; .7 teflon dielectric) **Note**: to specify the electrical delay for reflection measurements (in both directions), double the velocity factor.
Return Type	Double
Default	1
Examples	app.VelocityFactor = .66 'Write RelVel = app.VelocityFactor 'Read
C++ Syntax	HRESULT get_VelocityFactor(double *pVal) HRESULT put_VelocityFactor(double newVal)
Interface	IApplication

Write/Read

Visible Property

Description	Makes the Network Analyzer application visible or not visible. In the Not Visible state, the analyzer cycle time for making measurements can be significantly faster because the display does not process data.
VB Syntax	*app.***Visible** = *state*

Variable	**(Type) - Description**
app	An Application **(object)**
state	**(boolean)** **0** - Network Analyzer application **NOT** visible **1** - Network Analyzer application **IS** visible
Return Type	Boolean 0 - Not visible 1 - visible
Default	1

Examples

```
app.Visible = 0 'Write
vis = app.Visible 'Read
```

C++ Syntax	HRESULT get_Visible(VARIANT_BOOL * bVisible) HRESULT put_Visible(VARIANT_BOOL bVisible)
Interface	IApplication

About Analyzer Events

OnCalEvent

Description	Triggered by a calibration event. See a list of CAL Events. **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app_***OnCalEvent**(ByVal *eventID* As Variant, ByVal *chanNum* As Variant, ByVal *measNum* As Variant)

Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
chanNum	Channel Number of the event
measNum	Measurement Number of the event
Return Type	Not Applicable
Default	Not Applicable

Examples

```
Sub pna_OnCalEvent(ByVal eventID As Variant, ByVal channelNumber As Variant, ByVal measurementNumber As Variant)
'
MsgBox ("A Calibration event has occured")
End Sub
```

C++ Syntax	HRESULT OnCalEvent(VARIANT eventID, VARIANT channelNumber, VARIANT measurementNumber)
Interface	IApplication

OnChannelEvent

Description	Triggered by a channel event. See a list of Channel Events **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app_*__OnChannelEvent__(ByVal *eventID* As Variant, ByVal *chanNum* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
chanNum	Channel Number of the event
Return Type	Not Applicable
Default	Not Applicable
Examples	Sub pna_OnChannelEvent(ByVal eventID As Variant, ByVal channelNumber As Variant) MsgBox "A channel event occured" End Sub
C++ Syntax	HRESULT OnChannelEvent(VARIANT eventID, VARIANT channelNumber)
Interface	IApplication

About Analyzer Events

OnDisplayEvent

Description	Triggered by a display event. See a list of Display Events **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app_*__OnDisplayEvent__(ByVal *eventID* As Variant, ByVal *winNum* As Variant, ByVal *traceNum* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
winNum	Window Number of the event
traceNum	Trace Number of the event
Return Type	Not Applicable
Default	Not Applicable
Examples	Sub pna_OnDisplayEvent(ByVal eventID As Variant, ByVal windowNumber As Variant, ByVal traceNumber As Variant) MsgBox ("A Display event has occured") End Sub
C++ Syntax	HRESULT OnDisplayEvent(VARIANT eventID, VARIANT windowNumber, VARIANT traceNumber)
Interface	IApplication

OnHardwareEvent

Description	Triggered by a hardware event. See a list of Hardware Events **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app*_**OnHardwareEvent**(ByVal *eventID* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
Return Type	Not Applicable
Default	Not Applicable
Examples	Private Sub pna_OnHardwareEvent(ByVal eventID As Variant) MsgBox ("A Hardware event has occured") End Sub
C++ Syntax	HRESULT OnHardwareEvent(VARIANT eventID)
Interface	IApplication

OnMeasurementEvent

Description	Triggered by a measurement event. See a list of Measurement Events. **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app*_**OnMeasurementEvent**(ByVal *eventID* As Variant, ByVal *measNum* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
measNum	Measurement Number of the event
Return Type	Not Applicable
Default	Not Applicable
Examples	Private Sub pna_OnMeasurementEvent(ByVal eventID As Variant, ByVal measurementNumber As Variant) MsgBox ("A Measurement event has occured") End Sub
C++ Syntax	HRESULT OnMeasurementEvent(VARIANT eventID, VARIANT measurementNumber)
Interface	IApplication

About Analyzer Events

OnSCPIEvent

Description	Triggered by a SCPI event. See a list of SCPI Events **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app*_**OnSCPIEvent**(ByVal *eventID* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
Return Type	Not Applicable
Default	Not Applicable
Examples	Private Sub pna_OnSCPIEvent(ByVal eventID As Variant) MsgBox ("A SCPI event has occured") End Sub
C++ Syntax	HRESULT OnSCPIEvent(VARIANT eventID)
Interface	IApplication

About Analyzer Events

OnSystemEvent

Description	Triggered by a system event. See a list of System Events, also known as general events. **Note:** Some Severe Events are also used as Error Messages
VB Syntax	Sub *app*_**OnSystemEvent**(ByVal *eventID* As Variant)
Variable	**(Type) - Description**
app	An Application **(object)**
eventID	Code number of the event which occurred
chanNum	Channel Number of the event
Return Type	Not Applicable
Default	Not Applicable
Examples	Private Sub pna_OnSystemEvent(ByVal eventID As Variant) MsgBox ("A System event has occured") End Sub
C++ Syntax	HRESULT OnSystemEvent(VARIANT eventID)
Interface	IApplication

About Analyzer Events

OnUserEvent

Description	Reserved for future use.
VB Syntax	Sub app_**OnUserEvent**

Calibrator Object

Calibrator Object (default interface is ICalibrator)

See also Custom Interfaces

Calibrator Object Description

The Calibrator object is a child of the channel. It is a vehicle to perform calibration.

There must be a measurement present for the calibrator to use or you will receive an error (no measurement found). Therefore, to perform a 2-port cal, you must have any S-parameter measurement on the channel. For a 1-port measurement, you must have the measurement (S11 or S22) on the channel. The same is true for a response measurement.

New for Release 2.0 and greater:

Before you use the calibrator object to download or upload error terms, you must first specify the calibration type and ports that the calibration data applies to. This is because a Cal Set can hold more than one Cal Type and more than one combination of ports. So you must first do Calibrator.SetCalInfo (caltype, port1, port2)

Learn about reading and writing Calibration data.

There are a number of approaches to calibration with the calibrator object:

- You can collect data yourself and download it to the ACQUISITION buffer. The acquisition buffer holds the actual measured data for each standard. See the PNA data map.
    1. Calibrator.SetCalInfo
    2. Connect a standard
    3. Trigger a sweep
    4. Retrieve the data for the standard
    5. Download the data - calibrator.putStandard
    6. Repeat for each standard
    7. Calibrator.CalculateErrorCoefficients
- You can tell the calibrator to acquire a standard. In this case, the calibrator collects the data and places it in the ACQUISITION buffer.
    1. Calibrator.SetCalInfo
    2. Connect a standard
    3. Calibrator.AcquireCalStandard2
    4. Repeat for each standard
    5. Calibrator.CalcuateErrorCoefficients
- You can put previously-retrieved error terms in the error correction buffer.
    1. PutErrorTerm

2. Repeat for each term
3. Measurement.Caltype = pick one

- You can also "piece together" a 2-port cal from two 1-port cals (S11 and S22) and four response (thru) cals. The system will detect that all the standards needed for a 2-port cal have been acquired even though they may not have gathered at the same time.

Method	**Description**
AcquireCalConfidenceCheckECAL	Transfers ECAL confidence data into analyzer memory
AcquireCalStandard	Obsolete - use AcquireCalStandard2
AcquireCalStandard2	Causes the analyzer to measure a calibration standard. Also provides for sliding load.
CalculateErrorCoeffecients	Generates Error Terms from standard and actual data in the error correction buffer.
DoECAL1Port	Completes a 1 port ECAL
DoECAL2Port	Completes a 2 port ECAL
DoneCalConfidenceCheckECAL	Concludes an ECAL confidence check
GetECALModuleInfo	Returns information about the attached module
getErrorTerm	Obsolete - replaced by CalSet.getErrorTerm Retrieves error term data for the active calibration.
getStandard	Obsolete - replaced by CalSet.getStandard Retrieves calibration data from the aquisition data buffer (before error-terms are applied).
putErrorTerm	Obsolete - replaced by CalSet.putErrorTerm Puts error term data into the error-correction buffer for the active calibration.
putStandard	Obsolete - replaced by CalSet.putStandard Puts data into the acquisition data buffer (before error-terms are applied)
SaveCalSets	Writes new or changed Cal Sets out to disk. Shared with the CalManager Object
setCalInfo	Specifies the type of calibration and prepares the internal state for the rest of the calibration.
Property	**Description**
AcquisitionDirection	Specifies the direction in a 2-Port cal using one set of standards.
ECALCharacterization	Specifies which set of characterization data within an ECal module will be used for ECal operations with that module.
ECAL Isolation	Include Isolation in ECAL calibration
ECALPortMap	Specifies which ports of the ECal module are connected to which ports of the PNA for the DoECAL1Port and DoECAL2Port methods when the OrientECALModule property = False.
IsECALModuleFound	Tests communication between the PNA and ECAL Module
OrientECALModule	Specifies if the PNA should perform orientation of the ECal module during calibration.
Simultaneous2PortAcquisition	Allows the use of 2 sets of standards at the

same time.

Custom Interfaces

- **ICalData**
- **ICalibrator2**
- **ICalibrator3**

ICalData Interface

Description

Contains methods for putting Calibration data in and getting Calibration data out of the analyzer using typed data. This interface transfers data more efficiently than variant data.

Learn about reading and writing Calibration data.

Method	**Description**
getErrorTermComplex	Retrieves error term data
getStandardComplex	Retrieves calibration data from the acquisition data buffer (before error-terms are applied).
putErrorTermComplex	Puts error term data
putStandardComplex	Puts calibration data into the acquisition data buffer (before error-terms are applied).
Property	**Description**
None	

ICalibrator2 Interface

Description

The ICalibrator2 interface is supported by the Calibrator object. ICalibrator2 is derived from the Calibrator object's default interface -- ICalibrator. Therefore, ICalibrator2 supports all of the same methods and properties as ICalibrator.

ICalibrator2 also provides the additional methods and properties shown below.

Methods	**Description**
None	
Properties	**Description**
ECALCharacterization	Specifies which set of characterization data within an ECal module will be used for ECal operations with that module.

ICalibrator3_Interface

Description

This interface extends the Calibrator interface to expose port and orientation properties if ECAL

Learn about reading and writing Calibration data.

Methods	**Description**
None	
Properties	**Description**
ECALPortMap	Specifies which ports of the ECal module are connected to which ports of the PNA .
OrientECALModule	Specifies if the PNA should perform orientation of the ECal module during calibration.

Write-only **About ECAL Confidence Check**

AcquireCalConfidenceCheckECAL Method

Description	Transfers confidence data from the specified ECal module into the measurement's memory trace. The data is transferred to the specified S-parameter on the same channel as this Calibrator object. The characterization within the ECal module that the confidence data will be read from is specified by the **ECALCharacterization property** on the **ICalibrator2** interface. The default value of the ECALCharacterization property is **naECALFactoryCharacterization.**
VB Syntax	cal.AcquireCalConfidenceCheckECAL Sparam[,ecalModule]
Variable	**(Type) - Description**
cal	A Calibrator **(object)**
Sparam	**(String)** S-parameter to transfer confidence data to. This parameter must be present on the same channel as the calibrator object.
ecalModule	**(enum NAECALModule) –** Optional argument. ECal module. Choose from: 0 - naECALModule_A (default, if unspecified) 1 - naECALModule_B
Return Type	None
Default	Not applicable
Examples	Cal.AcquireCalConfidenceCheckECAL "S11", naECALModule_A
C++ Syntax	HRESULT AcquireCalConfidenceCheckECAL(_bstr_t strParameter, enum NAECALModule ecalModule);
Interface	ICalibrator

Write-only **About Calibration Standards**

AcquireCalStandard Method - Obsolete

Description	**Note**: This command has been replaced by AcquireCalStandard2 Method, which provides for acquisition of sliding load standards. All other

functionality is identical.

VB Syntax	*cal*.**AcquireCalStandard** *std[,index]*

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
std	**(enum NACalClass)** Standard to be measured. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru 5 - naSOLT_Isolation **TRL Standards** 1 - naTRL_Reflection 2 - naTRL_Line_Reflection 3 - naTRL_Line_Tracking 4 - naTRL_Thru 5 - naTRL_Isolation
index	**(long integer)** number of the standard. Optional argument - Used if there is more than one standard required to cover the necessary frequency

	range. If unspecified, value is set to 0.
Return Type	None
Default	Not Applicable

Examples	Cal.AcquireCalStandard naSOLT_Thru 'Write

C++ Syntax	HRESULT AcquireCalStandard(tagNACalClass enumClass, short standardNumber)
Interface	ICalibrator

Write-only **About Calibration Standards**

AcquireCalStandard2 Method

Description

Measures the specified standard from the selected calibration kit. The calibration kit is selected using app.CalKitType.

For 2-port calibration, it is also necessary to specify direction with AcquisitionDirection.

Note: To omit Isolation from a 2-port calibration, do not Acquire a cal standard for naSOLT_Isolation

Note: This command replaces AcquireCalStandard. This command provides for the acquisition of a sliding load cal. All other functionality is identical.

VB Syntax *cal.***AcquireCalStandard** *std[,index],slide*

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
std	**(enum NACalClass)** Standard to be measured. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short

3 - naSOLT_Load

4 - naSOLT_Thru

5 - naSOLT_Isolation

TRL Standards

1 - naTRL_Reflection

2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

index	**(long integer)** number of the standard. Optional argument - Used if there is more than one standard required to cover the necessary frequency range. If unspecified, value is set to 0.
slide	(**enum** as NACalStandardSlidingState) State of the sliding load. The slide should be set a minimum of four times. Seven is the maximum that can be stored. See an example of a sliding load cal. Choose from: 0 - **naNotSlidingStd** - not using a sliding load 1 - **naSlideIsSet** - slide is set for acquisition 2 - **naSlideIsDone** - this next acquisition will be the last. Calculations will then be performed.
Return Type	None
Default	Not Applicable
Examples	Cal.AcquireCalStandard naSOLT_Thru,naNotSlidingStd Cal.AcquireCalStandard naSOLT_Thru,2,naNotSlidingStd 'measures the second standard listed in the class of naSOLT_Thru
C++ Syntax	HRESULT AcquireCalStandard2(tagNACalClass enumClass,standardPosition, naNotSlidingStd, NACalStandardSlidingState slidingStandardState)
Interface	ICalibrator

Write-only **About Performing a Calibration**

CalculateErrorCoefficients Method

Description	This method is the final call in a calibration process. It calculates error-correction terms and turns error-correction ON.
VB Syntax	*cal*.**CalculateErrorCoeffecients**

Variable	**(Type) - Description**
cal	Calibrator **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	Cal.CalculateErrorCoeffecients
C++ Syntax	HRESULT CalculateErrorCoefficients()
Interface	ICalibrator

Write-only **About Calibration**

DoECAL1Port Method

Description	Does a 1-Port calibration using an ECAL module. You must first have a 1-port measurement active to perform the calibration. The characterization within the ECal module that will be used for the calibration is specified by the **ECALCharacterization property** on the **ICalibrator2** interface. The default value of the ECALCharacterization property is **naECALFactoryCharacterization**.
VB Syntax	cal.**DoECAL1Port** [port][,module]
Variable	**(Type) - Description**
cal	A Calibrator **(object)**
port	**(long integer)** Optional argument - Port number to calibrate. Choose from: 1 - Calibrate port 1 (default if unspecified) 2 - Calibrate port 2
module	**(enum NAEcalModule)** Optional argument - ECAL module. Choose from: 0 - naECALModule_A - (default if unspecified) 1 - naECALModule_B
Return Type	None
Default	Not Applicable
Examples	cal.DoECAL1Port,2,naECALModule_B
C++ Syntax	HRESULT DoECAL1Port(long port, tagNAECALModule ecalModule);
Interface	ICalibrator

Write-only **About Calibration**

DoECAL2Port Method

Description	Does a 2-Port calibration using an ECAL module. You must first have a 2-port measurement active to perform the calibration. The characterization within the ECal module that will be used for the calibration is specified by the **ECALCharacterization property** on the **ICalibrator2** interface. The default value of the ECALCharacterization property is **naECALFactoryCharacterization**.
VB Syntax	*cal*.**DoECAL2Port***[portA][,portB][,module]*

Variable	(Type) - Description
cal	A Calibrator **(object)**
portA	**(long integer)** Optional argument - Number of the receive port to calibrate. Choose from: 1 - Calibrate port 1 (default, if unspecified) 2 - Calibrate port 2 3 - Calibrate port 3 (if the PNA has 3 ports)
portB	**(long integer)** Optional argument - Number of the source port to calibrate. Choose from: 1 - Calibrate port 1 (default, if unspecified) 2 - Calibrate port 2 3 - Calibrate port 3 (if the PNA has 3 ports)
module	**(enum NAECALModule)** – Optional argument. ECal module. Choose from: 0 - naECALModule_A (default, if unspecified) 1 - naECALModule_B
Return Type	None
Default	Not Applicable

Examples *cal*.DoECAL2Port,1,2,naECALModule_B

C++ Syntax HRESULT DoECAL2Port(long rcvport, long srcPort, tagNAECALModule ecalModule);

Interface ICalibrator

Write-only **About ECAL Confidence Check**

DoneCalConfidenceCheckECAL Method

Description Concludes the Confidence Check and sets the ECal module back into the idle state.

VB Syntax *cal*.**DoneCalConfidenceCheckECAL**

Variable	(Type) - Description
cal	A Calibrator **(object)**
Return Type	None
Default	None

Examples *cal*.DoneCalConfidenceCheckECAL

C++ Syntax HRESULT DoneCalConfidenceCheckECAL( );

Interface ICalibrator

Write/Read

ECALCharacterization Property

Description	Specifies which set of characterization data within an ECal module will be used for ECal operations with that module. A user characterization is entered into a module using the ECal User Characterization feature on the PNA. . If this COM property is set to one of the values **naECALUserCharacterization1** through **naECALUserCharacterization5** for a particular module, and that module does not have a characterization corresponding to that user number, attempts to use that module will return an error until the property is set back to **naECALFactoryCharacterization**.
VB Syntax	cal.**ECALCharacterization**(module) = value
Variable	**(Type) - Description**
cal	A Calibrator **(object)**
module	**(enum NAECALModule)** – ECal module. Choose from: 0 - naECALModule_A 1 - naECALModule_B
value	**(enum NAECALCharacterization)** – Characterization data within the ECal module to be used for ECal operations. Choose from: 0 – naECALFactoryCharacterization 1 – naECALUserCharacterization1 2 – naECALUserCharacterization2 3 – naECALUserCharacterization3 4 – naECALUserCharacterization4 5 – naECALUserCharacterization5
Return Type	enum NAECALCharacterization
Default	naECALFactoryCharacterization
Examples	Dim cal As Calibrator Dim eCharacterization As NAECALCharacterization Set cal = PNAapp.ActiveChannel.Calibrator cal.ECALCharacterization = naECALUserCharacterization1 'Write eCharacterization = cal.ECALCharacterization 'Read
C++ Syntax	HRESULT put_ECALCharacterization(tagNAECALModule moduleNumber, tagNAECALCharacterization characterization); HRESULT get_ECALCharacterization(tagNAECALModule moduleNumber, tagNAECALCharacterization* characterization);
Interface	ICalibrator2

Read-only

GetECALModuleInfo Method

Description	Returns the following information about the connected ECAL module: model number, serial number, connector type, calibration date, min and max frequency. The characterization within the ECal module that this information will be

	read from is specified by the **ECALCharacterization property** on the **ICalibrator2** interface. The default value of the ECALCharacterization property is **naECALFactoryCharacterization.**
VB Syntax	*moduleInfo = cal*.**GetECALModuleInfo** *(module)*

Variable	**(Type) - Description**
moduleInfo	**(string)** - variable to store the module information
cal	A Calibrator **(object)**
module	**(enum NAECALModule)** – ECAL module. Choose from: 0 - naECALModule_A 1 - naECALModule_B
Return Type	String
Default	Not Applicable

Examples

```
info = cal.GetECALModuleInfo(naECALModule_A)
```

Example return string:

ModelNumber: 85092-60007, SerialNumber: 01386, ConnectorType: N5FN5F, PortAConnector: Type N (50) female, PortBConnector: Type N (50) female, MinFreq: 30000, MaxFreq: 9100000000, NumberOfPoints: 250, Calibrated: July 4 2002

C++ Syntax	HRESULT GetECALModuleInfo(tagNAECALModule ecalModule, BSTR* info);
Interface	ICalibrator

Read-only **About Measurement Calibration**

GetErrorTerm Method - Obsolete

Description

Note: This command is replaced by CalSet.getErrorTerm.

Retrieves error term data that is used for error correction. The data is complex pairs. Memory for the returned Variant is allocated by the server. The server returns a variant containing a two-dimensional safe Array.

This method returns a variant which is less efficient than getErrorTermComplex on the ICalData interface.

Learn about reading and writing Calibration data.

VB Syntax	*data = cal*.**getErrorTerm** *term, rcv. src*

Variable	**(Type) - Description**
data	Variant array to store the data.
cal	A Calibrator **(object)**
term	**(enum As NaErrorTerm).** Choose from: naErrorTerm_Directivity_Isolation naErrorTerm_Match naErrorTerm_Tracking
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port

To get this Error Term	**Specify these parameters:** *term*	*rcv*	*src*
Fwd Directivity	naET_Directivity Isolation	1	1

Rev Directivity	naET_Directivity Isolation	2	2
Fwd Isolation	naET_Directivity Isolation	2	1
Rev Isolation	naET_Directivity Isolation	1	2
Fwd Source Match	naErrorTerm_Match	1	1
Rev Source Match	naErrorTerm_Match	2	2
Fwd Load Match	naErrorTerm_Match	2	1
Rev Load Match	naErrorTerm_Match	1	2
Fwd Reflection Tracking	naErrorTerm_Tracking	1	1
Rev Reflection Tracking	naErrorTerm_Tracking	2	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1
Rev Trans Tracking	naErrorTerm_Tracking	1	2

Return Type Variant

Default Not Applicable

Examples

```
Dim varError As Variant
 varError = cal.getErrorTerm(naErrorTerm_Tracking,2,1)
```

C++ Syntax HRESULT getErrorTerm(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, VARIANT* pData)

Interface ICalibrator

Read-only **About Cal Sets**

GetStandard Method - Obsolete

Description

Note: This method has been replaced by calSet.getStandard.

Retrieves data that was acquired for a specific cal standard. This method returns the actual measurement data - not the calculated error terms.

This method returns a variant which is less efficient than getStandardComplex on the ICalData interface.

Learn about reading and writing Calibration data.

VB Syntax *data = cal.***getStandard***class,rcv,src*

Variable	**(Type) - Description**
data	Variant array to store the data.
cal	A Calibrator **(object)**
class	**(enum NACalClass)** Standard to be measured. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD

5 - naClassE

6 - naReferenceRatioLine

7 - naReferenceRatioThru

SOLT Standards

1 - naSOLT_Open

2 - naSOLT_Short

3 - naSOLT_Load

4 - naSOLT_Thru

5 - naSOLT_Isolation

TRL Standards

1 - naTRL_Reflection

2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
Return Type	**(variant)** - two-dimensional array (0:1, 0:NumberOfPoints-1)
Default	Not Applicable

Examples

```
Dim varStd As Variant
 varStd = cal.getStandard(naSOLT_Thru,2,1)
```

C++ Syntax

HRESULT raw_getStandard(tagNACalClass stdclass, long ReceivePort, long SourcePort, VARIANT* pData)

Interface ICalibrator

Write-only **About Measurement Calibration**

PutErrorTerm Method - Obsolete

Description	**Note**: This command is replaced by CalSet.putErrorTerm.
	Puts variant error term data into the error-correction buffer. See Accessing data.
	Learn about reading and writing Calibration data.
VB Syntax	*cal*.**putErrorTerm**(*term,rcv, src, data*)

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
term	**(enum As NaErrorTerm)** naErrorTerm_Directivity_Isolation naErrorTerm_Match naErrorTerm_Tracking
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
data	**(variant)** Error term data in a two-dimensional array (0:1, 0:numpts-1).

To get this Error Term	**Specify these parameters:** *term*	*rcv*	*src*
Fwd Directivity	naET_Directivity Isolation	1	1
Rev Directivity	naET_Directivity Isolation	2	2
Fwd Isolation	naET_Directivity Isolation	2	1
Rev Isolation	naET_Directivity Isolation	1	2
Fwd Source Match	naErrorTerm_Match	1	1
Rev Source Match	naErrorTerm_Match	2	2
Fwd Load Match	naErrorTerm_Match	2	1
Rev Load Match	naErrorTerm_Match	1	2
Fwd Reflection Tracking	naErrorTerm_Tracking	1	1
Rev Reflection Tracking	naErrorTerm_Tracking	2	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1
Rev Trans Tracking	naErrorTerm_Tracking	1	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1

Return Type	Not Applicable
Default	Not Applicable
Examples	Dim varError As Variant varError = cal.putErrorTerm (naErrorTerm_Tracking,2,1,VarData)
C++ Syntax	HRESULT putErrorTerm(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, VARIANT varData)
Interface	ICalibrator

Write-only

PutStandard Method - Obsolete

Description	**Note**: This command is replaced by CalSet.putStandard.
	Writes variant data to the error correction buffer holding measurement data acquired for a specific standard.
	Learn about reading and writing Calibration data.
VB Syntax	*cal*.**putStandard** *class,rcv,src,data*

Variable	(Type) - Description
cal	A Calibrator **(object)**
class	**(enum NACalClass)** Standard. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru 5 - naSOLT_Isolation **TRL Standards** 1 - naTRL_Reflection 2 - naTRL_Line_Reflection 3 - naTRL_Line_Tracking 4 - naTRL_Thru 5 - naTRL_Isolation
rcv	**(long)** - Receiver Port
src	**(long)** - Source Port
data	**(variant)** Two dimensional array (0:1, 0:points-1)
Return Type	Not Applicable

Default Not Applicable

Examples Dim varStd (1,200) As Variant
cal.putStandard naSOLT_Thru, 2, 1, varStd

C++ Syntax HRESULT raw_putStandard(tagNACalClass stdclass, long ReceivePort, long SourcePort, VARIANT varData)

Interface ICalibrator

Write-only **About Cal Sets**

SaveCalSets Method

Description

Writes new or changed Cal Sets out to disk. All Cal Sets are saved in a single file (PNACal Sets.dat). This file is updated at the following times:

- On Application Exit
- When a Cal Set has been deleted
- When a calibration has been performed through the front panel interface
- When this method is called

Call this method whenever the Cal Set data has been changed remotely. Learn more about reading and writing Cal data using COM

Note: There is also a Save method on the ICalSet interface. The difference is the following:
ICalSet::Save - saves the data for the current Cal Set to the disk.
ICalManager/Calibrator::SaveCalSets - saves every Cal Set that currently exists in the instrument to the disk.

VB Syntax *object.***SaveCalSets**

Variable	**(Type) - Description**
object	**(object) -** A CalManager object or a Calibrator object
Return Type	None
Default	Not Applicable

Example calMgr.SaveCalSets

C++ Syntax HRESULT SaveCalSets( );

Interface ICalManager
ICalibrator

Write-only **About Performing a Calibration**

SetCalInfo Method

Description

Specifies the type of calibration. This method should be the first method called on the calibrator object. It prepares the internal state for the rest of the calibration. Learn more about reading and writing Cal data using COM

VB Syntax *cal.***SetCalInfo(***type,rcvPort,srcPort***)**

Variable	**(Type) - Description**
cal	A Calibrator **(object)**

type	**(enum NACalType)** - Calibration type. Choose from: **0** - naCalType_Response_Open **1** - naCalType_Response_Short **2** - naCalType_Response_Thru **3** - naCalType_Response_Thru_And_Isol **4** - naCalType_OnePort **5** - naCalType_TwoPort_SOLT **6** - naCalType_TwoPort_TRL **7 -** naCalType_None **8** - naCalType_ThreePort_SOLT
	Note: The analyzer can measure both ports simultaneously, assuming you have two of each standard type. For a 2-port cal, See cal.Simultaneous2PortAcquisition
	Note: For 1-port cals, the source port = receiver port. For 2-port SOLT and TRL, it doesn't matter which port is specified as source and receiver
rcvPort	**(long integer)** - Receiver Port
srcPort	**(long integer)** - Source Port
Return Type	NACalType
Default	**7- naCalType_None**
Examples	cal.setCalInfo(naCalType_Response_Open,1,1)
C++ Syntax	HRESULT SetCalInfo(tagNACalType calType,long portA, long portB)
Interface	ICalibrator

Read / Write **About Performing a Calibration**

AcquisitionDirection Property

Description	Specifies the direction of each part of a 2-port calibration.
VB Syntax	*cal*.**AcquisitionDirection** = *value*
Variable	**(Type) - Description**
cal	A Calibrator **(object)**
value	**(enum NADirection)** - Choose from: **0 - naForward** - measures the forward direction **1 - naReverse** - measures the reverse direction
Return Type	Long Integer
Default	naForward
Examples	cal.AcquisitionDirection = naForward
C++ Syntax	HRESULT AcquisitionDirection(tagNADirection dir);
Interface	ICalibrator

Read/Write **About ECAL**

ECALIsolation Property

Description	Specifies whether the acquisition of the ECal calibration should include

	isolation or not.
VB Syntax	*cal*.ECALIsolation=*value*

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
value	**(boolean)** **False (0)** - Exclude Isolation **True (1)** - Include Isolation
Return Type	Boolean
Default	False (0)

Examples

```
Dim oPNA as AgilentPNA835x.Application
Dim oCal as Calibrator
Set oPNA = CreateObject("AgilentPNA835x.Application",
"MachineName")
Set oCal = oPNA.ActiveChannel.Calibrator
' Uncomment the following line to have the cal include
isolation
' oCal.ECALIsolation = True
' Uncomment the following line to have the cal omit
isolation
'oCal.ECALIsolation = False
oCal.DoECAL2Port ' Do the cal
```

C++ Syntax	void PutECALIsolation (VARIANT_BOOL bIsolationState); VARIANT_BOOL GetECALIsolation( );
Interface	Calibrator

Write/Read

ECALPortMap Property

Description	Specifies which ports of the ECal module are connected to which ports of the PNA for the DoECAL1Port and DoECAL2Port methods when the OrientECALModule property = False.
VB Syntax	*cal*.**ECALPortMap** = *value*

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
value	**(string)** This parameter is expected to be formatted in the following manner: ax,by,…zz where a, b and z are ports on the module (i.e., A and B on 2-port ECal modules), and x, y and z are PNA port numbers (i.e., 1 and 2 on a 2-port PNA). Ports of the module which are not being used for calibration should be omitted from the string. For example, if we had a 4-port ECal module with port A connected to PNA port 2, port B to PNA port 3, port C not connected, and port D to PNA port 1, the string would be: A2,B3,D1 DoECAL1Port or DoECAL2Port methods will fail if the port numbers passed to those methods are not in the string of this property and

	OrientECALModule property = False.
Return Type	String
Default	Not Applicable

Examples

Dim cal As Calibrator

Dim sPortMap As String

Set cal = PNAapp.ActiveChannel.Calibrator

cal.ECALPortMap = “a2,b1” 'Write

sPortMap = cal.ECALPortMap 'Read

C++ Syntax

HRESULT put_ECALPortMap(tagNAECALModule ecalModule, BSTR strPortMap);

HRESULT get_ECALPortMap(tagNAECALModule ecalModule, BSTR *strPortMap);

Interface ICalibrator3

Read only **About ECAL**

IsECALModuleFound Property

Description	Tests communication between the PNA and the specified ECal module.
VB Syntax	*moduleFound* = *cal*.**IsECALModuleFound** *(module)*

Variable	**(Type) - Description**
moduleFound	**(boolean) -** Variable to store the returned test result. **True -** The PNA identified the presence of the specified ECal module. **False** - The PNA did NOT identify the presence of the specified ECal module.
cal	**(object) -** A Calibrator object
module	**(enum NAECALModule) –** ECAL module. Choose from: 0 - naECALModule_A 1 - naECALModule_B
Return Type	Boolean
Default	Not applicable

Examples

```
Set cal = pna.ActiveChannel.Calibrator
moduleFound = cal.IsECALModuleFound(naECALModule_A)
```

C++ Syntax

HRESULT get_IsECALModuleFound(tagNAECALModule moduleNumber, VARIANT_BOOL *bModuleFound);

Interface Calibrator

Write/Read

OrientECALModule Property

Description	Specifies if the PNA should perform orientation of the ECal module during calibration. Orientation is a technique by which the PNA automatically determines which ports of the module are connected to which ports of the PNA. Orientation begins to fail at very low power levels or if there is much attenuation in the path between the PNA and the ECal module.
VB Syntax	cal.OrientECALModule = value

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
value	**(boolean)** **False (0)** – DoECAL1Port and DoECAL2Port methods will use the value of the ECALPortMap property to determine the port connections. **True (1)** - DoECAL1Port and DoECAL2Port methods will use Orientation technique to determine port connections.
Return Type	Boolean
Default	True (1)

Examples

```
Dim cal As Calibrator
Dim bOrient As Boolean
Set cal = PNAapp.ActiveChannel.Calibrator
cal.OrientECALModule = False 'Write
bOrient = cal.OrientECALModule 'Read
```

C++ Syntax

```
HRESULT put_OrientECALModule(VARIANT_BOOL bOrient);
HRESULT get_OrientECALModule(VARIANT_BOOL *bOrient);
```

Interface ICalibrator3

Read / Write **About Performing a Calibration**

Simultaneous2PortAcquisition Property

Description	Specifies whether a 2-port calibration will be done with a single set of standards (one port at a time) or with two sets of standards (simultaneously).
VB Syntax	*cal*.**Simultaneous2PortAcquisition** = *state*

Variable	**(Type) - Description**
cal	A Calibrator **(object)**
state	**(boolean)** - Choose from: **True** - measures 2 ports simultaneously **False** - measures 1 port at a time
Return Type	Boolean
Default	False

Examples

```
cal.Simultaneous2PortAcquisition = True
```

C++ Syntax

```
HRESULT put_Simultaneous2PortAcquisition( VARIANT_BOOL bTwoSetsOfStandards)
HRESULT Simultaneous2PortAcquisition( VARIANT_BOOL *bTwoSetsOfStandards)
```

Interface ICalibrator

Calibrator Object custom ICalData Interface

Read-only **About Accessing Data**

GetErrorTermComplex Method

Description	Retrieves error term data from the error correction buffer. The data is in complex pairs. Learn more about reading and writing Cal Data using COM. **Note:** This method exists on a non-default interface. If you cannot access this method, use the GetErrorTerm Method on ICalibrator.
VB Syntax	*eData*.**GetErrorTermComplex** *term, rcv, src, numPts, real(), imag()*

Variable	**(Type) - Description**
eData	An ICalData pointer to the Calibrator object
term	**(enum NAErrorTerm)** - The error term to be retrieved. Choose from: • **naErrorTerm_Directivity_Isolation** • **naErrorTerm_Match** **naErrorTerm_Tracking**
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
numPts	**(long integer)** - on input, max number of data points to return; on output: indicates the actual number of data points returned.
real()	**(single)** - array to accept the **real** part of the error-term. One-dimensional for the number of data points.
imag()	**(single)** - array to accept the **imaginary** part of the error-term. One-dimensional for the number of data points.

To get this	**Specify these parameters:**		
Error Term	*term*	*rcv*	*src*
Fwd Directivity	naET_Directivity Isolation	1	1
Rev Directivity	naET_Directivity Isolation	2	2
Fwd Isolation	naET_Directivity Isolation	2	1
Rev Isolation	naET_Directivity Isolation	1	2
Fwd Source Match	naErrorTerm_Match	1	1
Rev Source Match	naErrorTerm_Match	2	2
Fwd Load Match	naErrorTerm_Match	2	1
Rev Load Match	naErrorTerm_Match	1	2
Fwd Reflection Tracking	naErrorTerm_Tracking	1	1
Rev Reflection Tracking	naErrorTerm_Tracking	2	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1
Rev Trans Tracking	naErrorTerm_Tracking	1	2

Return Type	Single
Default	Not Applicable

Examples

```
ReDim rel(numpts)
 ReDim img(numpts)
```

```
Dim eData As ICalData
Set eData = chan.Calibrator
eData.getErrorTermComplex naErrorTerm_Directivity_Isolation, 1, 1, 201,
rel(0), img(0)
```

C++ Syntax HRESULT raw_getErrorTermComplex(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)

Interface ICalData

Write-only **About Cal Sets**

GetStandardComplex Method

Description Queries standards acquisition data from the Cal Set. The data is in complex pairs. Learn more about reading and writing Cal Data using COM.

Before calling this method from the ICalData2 interface you must open the Cal Set with OpenCal Set. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

Note: This method exists on a non-default interface. If you cannot access this method, use the GetStandard Method on ICal Set

VB Syntax *interface*.**getStandardComplex** *class, rcv, src, numPts, real(), imag()*

Variable **(Type) - Description**

interface An ICalData pointer to the Calibrator object **or**
An ICalData2 pointer to the Cal Set object(preferrred)

class **(enum NACalClass)** Standard to be measured. Choose from:

1 - naClassA

2 - naClassB

3 - naClassC

4 - naClassD

5 - naClassE

6 - naReferenceRatioLine

7 - naReferenceRatioThru

SOLT Standards

1 - naSOLT_Open

2 - naSOLT_Short

3 - naSOLT_Load

4 - naSOLT_Thru

5 - naSOLT_Isolation

TRL Standards

1 - naTRL_Reflection

2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
numPts	**(long integer)** - on input, max number of data points to return; on output: indicates the actual number of data points returned.
real()	**(single)** - array to accept the real part of the calibration data. One-dimensional for the number of data points.
imag()	**(single)** - array to accept the imaginary part of the calibration data. One-dimensional for the number of data points.
Return Type	**(single)**
Default	Not Applicable

Examples

```
Dim numpts as long
 numpts = ActiveChannel.NumberOfPoints
 ReDim r(numpts)  ' real part
 ReDim i(numpts)  ' imaginary part
 Dim Cal Set as Cal Set
 set Cal Set = pna.GetCalManager.GetCal SetByGUID( txtGUID )
 Dim sData As ICalData2
 Set sData = Cal Set
 sdata.getStandardComplex naSOLT_Open, 1, 1, numpts, r(0), i(0)
```

C++ Syntax HRESULT getStandardComplex(tagNACalClass stdclass, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)

Interface ICalData2

Write-only **About Accessing Data**

PutErrorTermComplex Method

Description Puts error term data into the error-correction data buffer. Learn more about

	reading and writing Cal data using COM
VB Syntax	*data*.**putErrorTermComplex** *term, rcv, src, numPts, real(), imag()*

Variable — **(Type) - Description**

data — An ICalData pointer to the Calibrator object

term — **(enum NAErrorTerm)** - The error term to be retrieved. Choose from:

- **naErrorTerm_Directivity_Isolation**
- **naErrorTerm_Match**

naErrorTerm_Tracking

rcv — **(long integer)** - Receiver Port

src — **(long integer)** - Source Port

numPts — **(long integer) -** number of data points in the array

real() — **(single)** - array containing the **real** part of the calibration data. One-dimensional: the number of data points.

imag() — **(single)** - array containing the **imaginary** part of the calibration data. One-dimensional: the number of data points.

To get this Error Term	**Specify these parameters:** *term*	*rcv*	*src*
Fwd Directivity	naET_Directivity Isolation	1	1
Rev Directivity	naET_Directivity Isolation	2	2
Fwd Isolation	naET_Directivity Isolation	2	1
Rev Isolation	naET_Directivity Isolation	1	2
Fwd Source Match	naErrorTerm_Match	1	1
Rev Source Match	naErrorTerm_Match	2	2
Fwd Load Match	naErrorTerm_Match	2	1
Rev Load Match	naErrorTerm_Match	1	2
Fwd Reflection Tracking	naErrorTerm_Tracking	1	1
Rev Reflection Tracking	naErrorTerm_Tracking	2	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1
Rev Trans Tracking	naErrorTerm_Tracking	1	2
Fwd Trans Tracking	naErrorTerm_Tracking	2	1

Return Type — Not Applicable

Default — Not Applicable

Examples

```
Dim eData As ICalData
 Set eData = chan.Calibrator
 eData.putErrorTermComplex naErrorTerm_Directivity_Isolation, 1, 1, 201,
rel(0), img(0)
```

C++ Syntax — HRESULT putErrorTermComplex(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)

Interface — ICalData

PutStandardComplex Method

Description

Puts standards acquisition data into the Cal Set. Learn more about reading and writing Cal data using COM

Before calling this method you must open the Cal Set with OpenCal Set. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

VB Syntax

interface.**putStandardComplex** *class, rcv, src, numPts,real(),imag()*

Variable	**(Type) - Description**
interface	A ICalData pointer to the Calibrator object **or** A ICalData2 pointer to the Cal Set object
class	**(enum NACalClass)** Standard. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru 5 - naSOLT_Isolation **TRL Standards** 1 - naTRL_Reflection 2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
numPts	**(long integer)** - number of data points in the arrays being sent.
real()	**(single)** - one-dimensional array containing the **real** part of the acquisition data. (0:points-1)
imag()	**(single)** - one-dimensional array containing the **imaginary** part of the acquisition data. (0:points-1)
Return Type	Not Applicable
Default	Not Applicable

Examples

```
Dim sdata As ICalData2
 Set sdata = calmanager.CreateCal Set( 1 )
 sdata.putStandardComplex naSOLT_Open, 1, 1, numpts, rel(0), img(0)
```

C++ Syntax

HRESULT putStandardComplex(tagNACalClass stdclass, long ReceivePort, long SourcePort, long lNumValues, float* pReal, float* pImag)

Interface ICalData
 ICal Set

CalFactorSegments Collection

CalFactorSegments Collection

Description

A collection object that provides a mechanism for iterating through the segments of a power sensor cal factor table. For more information about collections, see Collections in the Analyzer.

Methods	**Description**
Add	Adds a PowerSensorCalFactorSegment object to the collection
Item	Use to get a handle to a PowerSensorCalFactorSegment object in the collection.
Remove	Removes an object from the collection.
Properties	**Description**
Count	Returns the number of objects in the collection.
Parent	Returns a handle to the Parent object (PowerSensor) of this collection.

Write-only **About Source Power Cal**

Add (PowerSensorCalFactorSegment) Method

Description	Adds a PowerSensorCalFactorSegment to the CalFactorSegments collection. To ensure predictable results, it is best to remove all segments before defining a new list of segments. For each segment in the collection, do a seg.Remove.
VB Syntax	*segs*.**Add** *(item [size])*
Variable	**(Type) - Description**
segs	**(object) -** A CalFactorSegments collection (object)
item	**(variant)** - Number of the new segment. If it already exists, a new segment is inserted at the requested position.
size	**(long integer) -** Optional argument. The number of segments to add, starting with item. If unspecified, value is set to 1.
Return Type	None
Default	Not Applicable
Examples	segs.Add 1, 4 'Adds segments 1,2,3 and 4
C++ Syntax	HRESULT Add(VARIANT index, long size);
Interface	ICalFactorSegments

Cal Set Object

CalSet Object (default interface is ICalSet2)

Description

Use this interface to query and or change the contents of a Cal Set.

ICalSet2 is the **default** interface. It extends the ICalSet Interface by providing methods for transmitting string and List2 data to the Cal Set.

See also the **custom interfaces:**

ICalData2 for transmitting data to and from the Cal Set, avoiding using variants.

ICalData3 for transmitting **string** data to and from the Cal Set, avoiding using variants.

Learn about reading and writing Calibration data.

Methods	**Interface**	**Description**
CloseCalSet	ICalSet	Resets the CalType and port associations made in the OpenCal Set.
ComputeErrorTerms	ICalSet	Computes error terms for the CalType specified by a preceding OpenCal Set call.
Copy	ICalSet	Creates a new Cal Set and copies the current Cal Set data into it.

getErrorTerm	ICalSet	Retrieves variant error term data.
getErrorTermByString	**ICalSet2**	Returns the data array for a specific error term from the calset.
GetErrorTermList	ICalSet	Returns a list of error terms for the CalType specified by OpenCal Set
getErrorTermList2	**ICalSet2**	Returns a list of error term names found in a calset, containing the specified prefix.
GetGUID	ICalSet	Returns the GUID identifying a Cal Set
getStandard	ICalSet	Retrieves variant data that was acquired for a specific cal standard.
getStandardByString	**ICalSet2**	Returns the data for a specified standard.
GetStandardsList	ICalSet	Returns a list of standards required for CalType specified by OpenCal Set
getStandardList2	**ICalSet2**	Returns a list of standards contained by this calset for the specified caltype.
HasCalType	ICalSet	Verifies that the Cal Set object contains the error terms required to apply the specified CalType to an appropriate measurement.
OpenCalSet	ICalSet	Opens the set and restricts access to a set of Error Terms.
putErrorTerm	ICalSet	Writes variant error term data into the error-correction buffer.
putErrorTermByString	**ICalSet2**	Writes a name/array pair to there calset.
putStandard	ICalSet	Writes variant data that was acquired for a specific cal standard.
putStandardByString	**ICalSet2**	Writes a name/array pair to there calset.
Save	ICalSet	Saves the current Cal Set to PNACalSets.dat.
StringToNACalClass	ICalSet	Converts string values from GetStandardsList into enumeration data
StringToNAErrorTerm2	ICalSet	Converts string values from GetErrorTermList into enumeration data
Properties		**Description**
Description	ICalSet	Descriptive string assigned to the Cal Set

ICalData2 Interface

Description

Use this interface as an alternative to the ICalSet Interface when transmitting data to and from the Cal Set to avoid using variants.

Learn about reading and writing Calibration data.

Methods	**Description**
getErrorTermComplex	Retrieves complex error term data from the error correction buffer
getStandardComplex	Retrieves complex data from the error correction buffer
putErrorTermComplex	Writes complex error term data into the error correction buffer
putStandardComplex	Writes complex data to the error correction buffer
Properties	**Description**

None

ICalData3 Interface

Description

Use this interface as an alternative to the ICalSet Interface when transmitting string data to and from the Cal Set to avoid using variants.

Learn about reading and writing Calibration data.

Methods	Description
getErrorTermComplexByString	Returns data for a specific error term.
getStandardComplexByString	Returns data for a specific error term.
putErrorTermComplexByString	Writes a name/array pair to the calset
putStandardComplexByString	Writes a name/array pair to the calset
Properties	**Description**
None	

Write-only **About Cal Sets**

Close CalSet Method

Description	Closes read/write access to the Cal Set. See OpenCalSet for an explanation of gaining access to the Cal Set. When you are finished reading and writing data from or to the Cal Set, close the Cal Set. Subsequent read/writes will require a new OpenCal Set call. Reading and writing Cal Set data is performed with the PutStandard, GetStandard, PutErrorTerm, GetErrorTerm method calls. These methods are provided by the ICal Set and ICalData2 interfaces.
VB Syntax	*CalSet*.**CloseCalSet**
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
Return Type	Not Applicable
Default	Not Applicable
Examples	Cal Set.CloseCalSet
C++ Syntax	HRESULT CloseCalSet
Interface	ICalSet

Write-only **About Cal Sets**

ComputeErrorTerms Method

Description	Computes error terms for the caltype specified by a preceding OpenCal Set call. The Cal Set must first be opened using OpenCalSet. If this call has not been made, the following error is issued: E_NA_Cal Set_ACCESS_DENIED The standards data required for the CalType must be available in the Cal Set or this error will be returned: E_NA_STANDARD_NOT_FOUND. **Note:** Error term computation requires data for the actual calibration kit standards from the current kit definition. ComputeErrorTerms assumes that the standards were acquired using only one standard per class.
VB Syntax	*CalSet*.**ComputeErrorTerms**
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
Return Type	Not Applicable
Default	Not Applicable
Examples	CalSet.ComputeErrorTerms
C++ Syntax	HRESULT ComputeErrorTerms( )
Interface	ICalSet

Write-only **About Cal Sets**

Copy Method

Description	Creates a new Cal Set and copies the current Cal Set data into it. Therefore, you now have a clone Cal Set with a different ID. Use this command to manipulate data on a Cal Set without corrupting the original cal data.
VB Syntax	*CalSet*.**Copy**
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
Return Type	Not Applicable
Default	Not Applicable

Examples

```
Dim mgr As CalManager
 Dim ocalset As CalSet
 Dim newcalset As CalSet
 Set mgr = pna.GetCalManager
 'Create a new (empty) Cal Set.
 Set ocalset = mgr.CreateCalSet(1)
 ocalset.Description = "original calset"
 pna.Channel(1).SelectCalSet ocalset.GetGUID, True

 'Launch the cal wizard and allow the user to perform
the calibration.
 If pna.LaunchCalWizard(False) Then
 'If the Launch returns true then the calibration
```

```
finished.
 ocalset.Save

 'Copy the Cal Set to the new one.
 Set newcalset = ocalset.Copy
 newcalset.Description = "copy of original calset"

 Else
 'If the cal doesn't finish, delete the old Cal Set
 'so it isn't taking up unnecessary memory.
 mgr.DeleteCalSet ocalset.GetGUID
 End If
```

As a result, the programmer can manipulate the data in the new Cal Set and always revert back to the old Cal Set as needed.

C++ Syntax HRESULT Copy(ICalSet** pCalSet);

Interface ICalSet

Read-only **About Cal Sets**

GetErrorTerm Method

Description

Queries data from the Cal Set that was acquired for a specific standard.

Learn more about reading and writing Cal Data using COM.

Before calling this method you must open the Cal Set with OpenCalSet.. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

The data is complex pairs. The server returns a variant containing a two-dimensional safe array. Memory for the returned Variant is allocated by the server and must be released by client.

Note: See also getErrorTermComplex on the ICalData2 interface to avoid using the variant data type.

VB Syntax *data = CalSet.***getErrorTerm** *setID, term, rcv, src*

Variable	**(Type) - Description**
data	Variant array to store the data.
CalSet	A Cal Set **(object)**
setID	(**long integer) –** specifies which error term set to read data from. (0 is the master set of eterms.) To get data from interpolated error terms, you must first determine if an interpolated set exists by calling the GetCalSetUsageInfo method. If it returns a number greater than 0 for the error term set ID, then the channel is currently using interpolated arrays. In this case, you can read from either the interpolated array (setID > 0) or the master array (setID = 0). **Note:**:Interpolated error terms are destroyed when no longer being used.
term	**(enum As NaErrorTerm2).** Choose from: 0 - naET_Directivity (rcv = src) 1 - naET_SourceMatch (rcv = src)

2 - naET_ReflectionTracking (rcv = src)

3 - naET_TransmissionTracking (rcv != src)

4 - naET_LoadMatch (rcv != src)

5 - naET_Isolation (rcv != src)

rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
Return Type	Variant
Default	Not Applicable
Examples	Dim varError As Variant varError = CalSet.getErrorTerm(0,naET_TransmissionTracking,2,1)
C++ Syntax	HRESULT getErrorTerm(long setID, tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, VARIANT* pData)
Interface	ICalSet

Read-only

GetErrorTermByString

Description	Returns the data array for a specific error term from the calset
VB Syntax	*pdata* **=** *Calset***.GetErrorTermByString**(*SetNumber*, *bufferName*)
Variable	**(Type) - Description**
calset	A Calset **(object)**
SetNumber	**(long integer)** SetNumber = 0 is the master set of error terms for this calset. Interpolated error terms reside in sets > 0. To determine what set number is being used by a channel, see **ICalManager::GetCalSetUsage**.
bufferName	**(String)** The string used to identify a particular buffer in the calset. An example string for port 3 directivity in a full 2 port cal might be "Full 2 Port Cal (2,3)::Directivity(3,3)".
pdata	**(Variant)** The pdata array is a variant. This data is usually two dimensional. Each element of the variant array is a 2 array of type single. The two elements represent the real and imaginary parts of a complex datum. **Note:** This structure is compatible with scripting clients who can only use variants. For alternative methods that use typed arrays, see ICalData3.
Return Type	Variant
Default	Not Applicable
Examples	See an example
C++ Syntax	**HRESULT GetErrorTermByString (long** SetNumber**, BSTR** bufferName, **VARIANT*** pdata**);**
Interface	ICalSet2

Write-only **About Cal Sets**

GetErrorTermList Method

Description

Returns the list of Error Terms contained in this Cal Set for the CalType specified in the OpenCal Set method. Learn more about reading and writing Cal Data using COM.

The list is a comma separated, textual representation of the error terms with the term name followed by the port path in parentheses:

Term (n, n),

Term (m,n)

Before calling this method you must open the Cal Set with OpenCal Set.. If the Cal set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

Use StringToNAErrorTerm2 to convert the list entrees to values that can be used with GetErrorTerm and PutErrorTerm.

Note: The port path designation (m n) indicates the ports that contribute to the error being compensated. Directivity, source match and reflection tracking are single port characteristics, designated in this list by (n n) where n equals the port being characterized.
Other terms characterize the interaction between ports. For example, the load match term is describing the match at port (m) while looking into port (n). Thus the notation (m n) indicates the two ports that contribute to the loadmatch error.

VB Syntax *CalSet*.**GetErrorTermList** (SetID, count, strList)

Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
SetID	(long) - specifies the error term set to query. Use 0 for the master set.
count	(long) - the number of error terms in the returned list
strList	**(string) -** comma separated list of error terms found in Cal Set
Return Type	Not Applicable
Default	Not Applicable

Examples

```
dim count as Integer
 dim list as string
 OpenCalSet (naCalType_TwoPortSOLT 1, 2)
 GetErrorTermList( 0, count, list)
 CloseCalSet( )
```

Assuming the cal set contained the full set of error terms for this two-port Cal, the returned list would be:

"Directivity(1 1),SourceMatch(1 1),ReflectionTracking(1 1),TransmissionTracking(2 1),LoadMatch(2 1),Isolation(2 1),Directivity(2 2),SourceMatch(2 2),ReflectionTracking(2 2),TransmissionTracking(1 2),LoadMatch(1 2),Isolation(1 2)"

C++ Syntax HRESULT GetErrorTermList (long etermSetID, long* count, BSTR* strList);

Interface ICalSet

Read-only

GetErrorTermList2

Description Returns a list of error terms names found in the calset, containing the

	specified prefix.
VB Syntax	**GetErrorTermList2**(*SetNumber*, *calTypePrefix*)

Variable	**(Type) - Description**
SetNumber	**(Long)** SetNumber = 0 is the master set of error terms for this caltype, in this calset. Interpolated error terms reside in sets > 0. To determine what set number is being used, see **ICalManager::GetCalSetUsage.**
caltypePrefix	**(String)** The string used to identify calset data as belonging to a specific caltype. (By convention, calset buffers are named "CalType:Item".) An example prefix for a two port cal on ports 2 and 3 might be: "Full 2 Port Cal (2,3)". If the prefix is empty, all terms are returned. This string is used as a filter so that only the buffer names of interest are returned.
list	**(Variant)** Variant containing a string array of error term names
Return Type	Variant
Default	Not Applicable

Examples	See the following example

C++ Syntax	**HRESULT GetErrorTermList2(long** SetNumber**, BSTR** caltypePrefix**, VARIANT*** list**)**
Interface	ICalSet2

Read-only **About Cal Sets**

GetGuid Method

Description	Returns a string containing the GUID identifying this Cal Set. Each Cal Set is assigned a GUID (global unique ID). GUIDs are used to retrieve and select Cal Sets on the PNA. Learn more about reading and writing Cal Data using COM.

VB Syntax	*value = CalSet.***GetGuid**
Variable	**(Type) - Description**
value	**(string)** - Variable to store the returned GUID
CalSet	**(object) -** A Cal Set object
Return Type	String
Default	Not Applicable

Examples	guid = CalSet.GetGuid 'Read

C++ Syntax	HRESULT GetGUID(BSTR* pGUIDString);
Interface	ICalSet

Read-only **About Cal Sets**

GetStandard Method

Description	Queries data from the Cal Set that was acquired for a specific standard. Learn more about reading and writing Cal Data using COM. Before calling this method you must open the Cal Set with OpenCal Set.. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

The data is complex pairs. The server returns a variant containing a two-dimensional safe array. Memory for the returned Variant is allocated by the server and must be released by client.

Note: See also getStandardComplex on the ICalData2 interface to avoid using the variant data type.

VB Syntax	*data = CalSet.***getStandard** *standard, rcv , src*

Variable	(Type) - Description
data	Variant array to store the data.
CalSet	A **Cal Set (object)**
standard	**(enum NACalClass)** Standard to be measured. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru 5 - naSOLT_Isolation **TRL Standards** 1 - naTRL_Reflection 2 - naTRL_Line_Reflection 3 - naTRL_Line_Tracking 4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
Return Type	**(variant)** - two-dimensional array (0:1, 0:NumberOfPoints-1)
Default	Not Applicable

Examples

```
Dim varStd As Variant
Dim varStd2 As Variant

Cal Set.OpenCalSet( naCalType_TwoPortSOLT, 1, 2)
 varStd = CalSet.getStandard(naSOLT_Thru,2,1)
varStd2 = Cal Set.getStandard(naSOLT_Thru,1,2)
```

Cal Set.CloseCalSet( )

C++ Syntax	HRESULT getStandard(tagNACalClass stdclass, long ReceivePort, long SourcePort, VARIANT* pData)
Interface	ICalSet

Read-only

GetStandardByString

Description	Returns the data for a specific standard.
VB Syntax	**GetStandardByString**(*bufferName, pdata*)
Variable	**(Type) - Description**
bufferName	**(String)** The string used to identify a particular buffer in the calset. An example string requesting the data for the Load standard in a full 2 port cal might be "Full 2 Port Cal (2,3)::S11C(3,3)".
pdata	**(Variant)** The pdata array is a variant. This data is usually two dimensional. Each element of the variant array is a 2 array of type single. The two elements represent the real and imaginary parts of a complex datum. **Note:** This structure is compatible with scripting clients who can only use variants. For alternative methods that use typed arrays, see ICalData3.
Return Type	Variant
Default	Not Applicable
Examples	See the following example
C++ Syntax	**HRESULT GetStandardByString(BSTR** bufferName**, VARIANT*** pdata**)**
Interface	ICalSet2

Read-only **About Cal Sets**

GetStandardsList Method

Description Returns the list of Standards contained in this Cal Set for the CalType specified in the OpenCal Set method. Learn more about reading and writing Cal Data using COM.

The list is a comma separated, textual representation of the error terms with the term name followed by the port path in parentheses.

Standard (n, n),

Standard (m, n)

Before calling this method you must open the Cal Set with OpenCal Set. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

Use StringToNACalClass to convert the list entrees to values that can be used with GetStandard and PutStandard.

Note: The port path designation (m n) indicates the receive and source ports for the measurement. Shorts, opens and loads are single port devices, designated in this list by (n n) where n equals the port to which the device is connected. These devices are all characterized by reflection measurements.
The dual port thru device is characterized by both transmission and reflection measurements in order to compensate for load match and tracking terms.
The notation (n n) indicates the reflection measurement for this device.
The notation (m n) indicates the transmission measurement, where the source and receive ports are different.

VB Syntax	*CalSet*.**GetStandardsList** *(count, list)*
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
count	**(long [out])** - indicates the number of items returned in the list
list	**(string)** - Variable to store the returned Comma separated list of items.
Return Type	String
Default	Not Applicable

Examples

```
dim count as Integer
 dim list as string
 OpenCalSet (naCalType_TwoPortSOLT, 1, 2)
 GetStandardsList( count, list)
 CloseCalSet( )
```

Assuming the Cal Set contained the full set of standards for this two port cal, the returned list would be:

```
"Open(1 1),
Short(1 1),
Load(1 1),
Thru(1 1),
Isolation(2 1),
Open(2 2),
Short(2 2),
Load(2 2),
Thru(2 2),
Isolation(1 2)
Thru(2 1),
Thru(1 2)"
```

C++ Syntax HRESULT GetStandardsList(long* count, BSTR* list);

Interface ICalSet

Read-only

GetStandardList2

Description	Returns a list of standards contained by this calset for the specified caltype.
VB Syntax	**GetStandardList2**(*calType, list*)
Variable	**(Type) - Description**
caltype	**(String)** The string used to identify calset data as belonging to a specific caltype. (By convention, calset buffers are named "CalType:Item"). An example string for a two port cal on ports 2 and 3 might be: "Full 2 Port Cal (2,3)". If the string is empty, all standards are returned. This string is used as a filter so that only the buffer names of interest are returned.
list	**(Variant)** Variant containing a string array of standards for the specified caltype.
Return Type	Variant
Default	Not Applicable
Examples	See the following example
C++ Syntax	**HRESULT GetErrorTermList2(BSTR** caltype**, VARIANT*** list**)**
Interface	ICalSet2

Read-only **About Cal Sets**

HasCalType Method

Description Verifies that the Cal Set object contains the error terms required to perform the specified correction (CalType) to an appropriate measurement.

The argument list includes specifiers for up to 3 ports. The number of arguments required depends on the CalType specified. The value for each port is set to 0 if not specified.

* order of port arguments is significant for these caltypes

Caltype	**Port arguments required**
naCalType_Response_Open	Port1
naCalType_Response_Short	Port1
*naCalType_Response_Thru	Port1 (rcv), Port2 (src)
*naCalType_Response_Thru_And_Isol	Port1 (rcv), Port2 (src)
naCalType_OnePort	Port1
naCalType_TwoPort_SOLT	Port1, Port2
naCalType_TwoPort_TRL	Port1, Port2
naCalType_ThreePort_SOLT	Port1, Port2, Port3

VB Syntax *check = CalSet.***HasCalType** *calType, port1, port2, port3*

Variable	(Type) - Description
check	**(boolean) -** variable to store the returned value **TRUE (nonzero) -** Cal Set has all of the error terms necessary to apply the specified correction (CalType) **FALSE(0) -** Cal Set DOES NOT have all of the error terms necessary to apply the specified CalType
CalSet	**(object) -** A Cal Set object
calType	(enum as naCalType) - type of correction to be applied. Choose from **0** - naCalType_Response_Open **1** - naCalType_Response_Short **2** - naCalType_Response_Thru **3** - naCalType_Response_Thru_And_Isol **4** - naCalType_OnePort **5** - naCalType_TwoPort_SOLT **6** - naCalType_TwoPort_TRL **7** - naCalType_None **8** - naCalType_ThreePort_SOLT
port1	**(long)** - required. This argument must be specified. This specifies either: - the one significant port for an open/short response cal or a 1 port cal. - or one of the ports involved in a 2 or 3 port cal - or the ***receive*** port for a thru response / thru-isolation cal.
port2	**(long) -** required for any caltype involving more than one port This specifies either: - one of the ports involved in a 2 or 3 port cal (order independent) - or the **source** port for a thru response / thru-isolation cal
port3	**(long) -** required only for 3 port cal This specifies either: - one of the ports involved in a 3 port cal (order independent)

Return Type VARIANT_BOOL

Default Not Applicable

Examples value = CalSet.HasCalType(naCalType_TwoPort_TRL, 1, 2)

C++ Syntax HRESULT HasCalType(tagNACalType, long port1, long port2, long port3, BOOL *pVal);

Interface ICalSet

Read-only **About Cal Sets**

OpenCalSet Method

Description Open the Cal Set to read/write a particular **CalType.** Learn more about reading and writing Cal Data using COM.

This method is a prerequisite to several other Cal Set methods.

A Cal Set can contain more than one **caltype.** This method opens the Cal Set and restrict access to a particular set of terms. Subsequent commands like PutErrorTerm and GetErrorTerm use this information to access the correct error terms in the Cal Set. For example:

`OpenCal Set( naCalType_TwoPortSOLT, 3, 2, 0)`

`PutErrorTerm( naDirectivity, 1, 1, Buffer)`

The directivity error term for port 1 could belong to any number of caltypes: Full1Port (S11), Full2Port (12), Full2Port (13) or Full3Port (123). The **CalType and port** specifiers in the OpenCalSet call direct the uploaded directivity term to the correct set of error terms.

To close the CalType, see CloseCalSet.

The argument list includes three port specifiers. The following table shows which of these arguments are significant, given the **CalType** specified.

Caltype	**Port arguments required**
naCalType_Response_Open	Port1
naCalType_Response_Short	Port1
*naCalType_Response_Thru	Port1 (rcv), Port2(src)
*naCalType_Response_Thru_And_Isol	Port1 (rcv), Port2(src)
naCalType_OnePort	Port1
naCalType_TwoPort_SOLT	Port1, Port2
naCalType_TwoPort_TRL	Port1, Port2
naCalType_ThreePort_SOLT	Port1, Port2, Port3

* order of port arguments is significant for these caltypes

VB Syntax	*CalSet*.**OpenCalSet** (CalType, port1, port2, port3)
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
CalType	(enum as naCalType) - type of correction to be applied. Choose from **0** - naCalType_Response_Open **1** - naCalType_Response_Short **2** - naCalType_Response_Thru **3** - naCalType_Response_Thru_And_Isol **4** - naCalType_OnePort **5** - naCalType_TwoPort_SOLT **6** - naCalType_TwoPort_TRL **7** - naCalType_None **8** - naCalType_ThreePort_SOLT
port1	**(long)** - required. This argument must be specified. This specifies either: - the one significant port for an open/short response cal or a 1 port cal. - or one of the ports involved in a 2 or 3 port cal - or the ***receive*** port for a thru response / thru-isolation cal.
port2	**(long) -** required for any caltype involving more than one port This specifies either: - one of the ports involved in a 2 or 3 port cal (order independent)

	- or the ***source*** port for a thru response / thru-isolation cal
port3	**(long) -** required only for 3 port cal This specifies either: - one of the ports involved in a 3 port cal (order independent)
Return Type	None
Default	Not Applicable
Examples	CalSet.OpenCalSet naCalType_ThreePort_SOLT, 3,2,1
C++ Syntax	HRESULT OpenCalSet (naCalType, port1, [optional] port2, [optional] port3);
Interface	ICalSet

Write-only **About Cal Sets**

PutErrorTerm Method

Description	Puts error term data into the Cal Set. Learn more about reading and writing Cal data using COM Before calling this method you must open the Cal Set with OpenCal Set.. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED. The data must be complex pairs, contained in a two-dimensional VARIANT array. **Note:** See also PutErrorTermComplex on the ICalData2 interface to avoid using the variant data type.
VB Syntax	*CalSet.***putErrorTerm** (*term, rcv, src, data*)
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
term	**(enum As NaErrorTerm2)** Error Term. Choose from: 0 - naET_Directivity (src = rcv) 1 - naET_SourceMatch (src = rcv) 2 - naET_ReflectionTracking (src = rcv) 3 - naET_TransmissionTracking (src != rcv) 4 - naET_LoadMatch (src != rcv) 5 - naET_Isolation (src != rcv)
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
data	**(variant)** Error term data in a two-dimensional array (0:1, 0:numpts-1).
Return Type	Not Applicable
Default	Not Applicable
Examples	Private Sub Form_Load() Set pna=CreateObject("AgilentPNA835x.Application") InitPhonyData PutPhonyData End Sub

```
Private Sub InitPhonyData()
Dim i
Dim numpts
numpts = ActiveChannel.NumberOfPoints
ReDim v(numpts - 1, 1)

For i = 0 To numpts - 1
v(i, 0) = i
v(i, 1) = 0
Next

End Sub

Private Sub PutPhonyData()
Dim cset As CalSet
Set cmgr = pna.GetCalManager
Set cset = cmgr.CreateCalSet(1)
cset.OpenCalSet naCalType_OnePort, 1
cset.putErrorTerm naET_Directivity, 1, 1, v
cset.putErrorTerm naET_ReflectionTracking, 1, 1, v
cset.putErrorTerm naET_SourceMatch, 1, 1, v
cset.CloseCalSet
cset.Description = "Phony One Port"
guid = cset.GetGUID

End Sub
```

C++ Syntax	HRESULT putErrorTerm(tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, VARIANT varData)
Interface	ICalSet

Write-only

PutErrorTermByString

Description	Writes a name/array pair to the calset
VB Syntax	**PutErrorTermByString**(*bufferName, vdata*)
Variable	**(Type) - Description**
bufferName	**(String)** The string used to specify a particular buffer in the calset. An example string for port 3 directivity in a full 2 port cal might be "Full 2 Port Cal (2,3)::Directivity(3,3)".
vdata	**(Variant)** The vdata array is a variant. This data is usually two dimensional. Each element of the variant array is a 2 array of type single. The two elements represent the real and imaginary parts of a complex datum. **Note:** This structure is compatible with scripting clients who can only use variants. For alternative methods that use typed arrays, see ICalData3.
Return Type	Not Applicable
Default	Not Applicable
Examples	see example
C++ Syntax	**HRESULT PutStandardByString(BSTR** bufferName**, VARIANT** vardata**)**

Interface ICalSet2

Write-only

PutErrorTermComplexByString

Description	Writes a name/array pair to the calset
VB Syntax	**PutErrorTermComplexByString**(bufferName, lnumPoints, real(0), imag(0))
Variable	**(Type) - Description**
bufferName	**(String)** The string used to identify a particular buffer in the calset. An example string for port 3 directivity in a full 2 port cal might be "Full 2 Port Cal (2,3)::Directivity(3,3)".
lnumpoints	**(Long)** The capacity of the float arrays in arguments "real" & "imag".
real	**(Single)** The real component of the complex data.
imag	**(Single)** The imaginary component of the complex data.
	Note: The size of the real and imaginary arrays should be the same.
Return Value	Single
Default	Not Applicable
Examples	See example
C++ Syntax	**HRESULT PutErrorTermComplexByString(BSTR** bufferName**,** **long** lnumPoints**,** **float*** real**,** **float*** imag**);**
Interface	ICalData3

Write-only **About Cal Sets**

PutStandard Method

Description

Puts data into the CalSet. Learn more about reading and writing Cal data using COM

Before calling this method you must open the Cal Set with OpenCal Set. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.

The data is complex pairs. The server returns a variant containing a two-dimensional safe array. Memory for the returned Variant is allocated by the server and must be released by client.

Note: See also PutStandardComplex on the ICalData2 interface to avoid using the variant data type.

VB Syntax	*obj*.**putStandard** *class, rcv, src, data*
Variable	**(Type) - Description**
obj	**(object) -** A Calibrator or Cal Set object
class	**(enum NACalClass)** Standard. Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru 5 - naSOLT_Isolation **TRL Standards** 1 - naTRL_Reflection 2 - naTRL_Line_Reflection 3 - naTRL_Line_Tracking 4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long)** - Receiver Port
src	**(long)** - Source Port
data	**(variant)** Two dimensional array (NUMPTS, 2)
Return Type	Not Applicable
Default	Not Applicable

Examples

```
Dim cmgr as CalManager
 Dim cset As CalSet
 Set cmgr = pna.GetCalManager
 Set cset = cmgr.CreateCalSet(1)
 cset.OpenCalSet naCalType_OnePort, 1
 cset.putStandard naSOLT_Open, 1, 1,
varOpen
 cset.putStandard naSOLT_Short, 1, 1,
varShort
 cset.putStandard naSOLT_Load, 1, 1,
varLoad
 cset.ComputeErrorTerms
 cset.CloseCalSet
 cset.Description = "Uploaded one
port cal"
 guid = cset.GetGUID

 End Sub
```

C++ Syntax	HRESULT putStandard(tagNACalClass stdclass, long ReceivePort, long SourcePort, VARIANT varData)
Interface	ICalibrator ICalSet

Write-only

PutStandardByString

Description	Writes a name array/pair to the calset.
VB Syntax	**PutStandardByString**(*bufferName*, *vdata*)
Variable	**(Type) - Description**
bufferName	**(String)** The string used to specify a particular buffer in the calset. An example string requesting the data for the Load standard in a full 2 port cal might be "Full 2 Port Cal (2,3)::S11C(3,3)".
vdata	**(Variant)** The variant containing a safearray of variants. This data is usually two dimensional. **Note:** The vardata array is a safearray of variants wrapped in a variant. This

	structure is compatible with scripting clients who can only use variants. For alternative methods that used typed arrays, see ICalData3.
Return Type	Not Applicable
Default	Not Applicable
Examples	see the example
C++ Syntax	**HRESULT PutStandardByString(BSTR** bufferName**, VARIANT** vardata**);**
Interface	ICalSet2

Write-only

PutStandardComplexByString

Description	Writes a name/array pair to the calset.
VB Syntax	**PutStandardComplexByString**(*bufferName, lnumPoints , real(o) , imag(0)*)
Variable	**(Type) - Description**
bufferName	**(String)** The string used to identify a particular buffer in the calset. An example string for port 3 directivity in a full 2 port cal might be "Full 2 Port Cal (2,3)::Directivity(3,3)".
lnumpoints	**(Long)** The size of the arrays in arguments 4 and 5.
real	**(Single)** The real component of the complex data.
imag	**(Single)** The imaginary component of the complex data.
	Note: The size of the real and imaginary arrays should be the same.
Return Value	Single
Default	Not Applicable
Examples	See example
C++ Syntax	**HRESULT PutStandardComplexByString(BSTR** bufferName**,** **long** lnumPoints**,** **float*** real**,** **float*** imag**);**
Interface	ICalData3

Write-only

About Cal Sets

Save Method

Description	Saves the current Cal Set to the PNACalSets.dat file. Learn more about reading and writing Cal data using COM **Note:** There is also a Save method on the ICalManager and Calibrator interface. The difference is the following: ICalSet::Save - saves the data for the current Cal Set to the disk. ICalManager/Calibrator::SaveCalSets - saves every Cal Set that currently exists in the instrument to the disk.
VB Syntax	*CalSet*.**Save**
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
Return Type	Not Applicable
Default	Not Applicable

Examples `mycalSet.Save`

See Copy Method for an example application of this command.

C++ Syntax HRESULT Save();

Interface ICalSet

Read-only **About Cal Sets**

StringToNACalClass Method

Description	Converts the returned strings from GetStandardsList into the enumeration (NACalClass) and the port numbers required for PutStandard and GetStandard methods that transmit data in and out of the Cal Set. Learn more about reading and writing Cal data using COM
VB Syntax	*CalSet*.**StringToNACalClass** (*list*, *std, rcv, src)*
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
list	**(string)** - a string containing the textual description of the standard.
std	**(enum NACalClass)** Choose from: 1 - naClassA 2 - naClassB 3 - naClassC 4 - naClassD 5 - naClassE 6 - naReferenceRatioLine 7 - naReferenceRatioThru **SOLT Standards** 1 - naSOLT_Open 2 - naSOLT_Short 3 - naSOLT_Load 4 - naSOLT_Thru

5 - naSOLT_Isolation

TRL Standards

1 - naTRL_Reflection

2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

rcv	**(long)** - port number of the receiver
src	**(long)** - port number of the source
Return Type	Not Applicable
Default	Not Applicable
Examples	guid = CalSet.StringToNACalClass(*list*, *std, rcv, src)*
C++ Syntax	HRESULT StringtoNACalClass (BSTR* str, NACalClass* item, long *rcv, long *src);
Interface	ICalSet

Read-only **About Cal Sets**

StringtoNAErrorTerm2 Method

Description	Converts the returned strings from GetErrorTermList into the enumeration (NAErrorTerm2) and the port numbers required for PutErrorTerm and GetErrorTerm methods that transmit data in and out of the Cal Set. Learn more about reading and writing Cal data using COM
VB Syntax	*Cal Set*.**StringToNAErrorTerm2** (*list, eterm, rcv, src*)
Variable	**(Type) - Description**
Cal Set	**(object) -** A Cal Set object
list	**(string)** - a string containing the textual description of the error term.
eterm	**(enum As NaErrorTerm2).** Choose from: 0 - naET_Directivity (rcv = src) 1 - naET_SourceMatch (rcv = src) 2 - naET_ReflectionTracking (rcv = src) 3 - naET_TransmissionTracking (rcv != src) 4 - naET_LoadMatch (rcv != src)

	5 - naET_Isolation (rcv != src)
rcv	**(long)** - port number of the receiver
src	**(long)** - port number of the source
Return Type	Not Applicable
Default	Not Applicable
Examples	CalSet.StringToNAErrorTerm2 str, term, rcv, src
C++ Syntax	HRESULT StringToNAErrorTerm2 (BSTR* str, NAErrorTerm2* item, long *rcv, long *src);
Interface	ICalSet

Write / Read **About Cal Sets**

Description Property

Description	Sets or returns the descriptive string assigned to the Cal Set. Change this string so that you can easily identify each Cal Set constructed.
VB Syntax	*CalSet*.**Description** = *value*
Variable	**(Type) - Description**
CalSet	**(object) -** A Cal Set object
value	**(string)** – Description of the Cal Set
Return Type	String
Default	"CalSet_n" where n is an integer number.
Examples	CalSet.Description = "My Cal Set" 'Write desc = CalSet.Description 'Read
C++ Syntax	HRESULT get_Description(BSTR *pVal) HRESULT put_Description(BSTR newVal);
Interface	ICalSet

Cal Set Object custom ICalData2 Interface

Read-only **About Cal Sets**

GetErrorTermComplex Method

Description	Queries error term data from the Cal Set. The data is in complex pairs. Learn more about reading and writing Cal Data using COM. **Note:** This method exists on a non-default interface. If you cannot access this method, use the GetErrorTerm Method on ICal Set.
VB Syntax	*eData.***GetErrorTermComplex** *setID, term, rcv, src, numPts, real(), imag()*
Variable	**(Type) - Description**
eData	An ICalData2 pointer to the Cal Set object
setID	(**long integer) –** specifies which error term set to read data from. (0 is the master set of eterms.) To get data from interpolated error terms, you must first determine

	if an interpolated set exists by calling the GetCalSetUsageInfo method. If it returns a number greater than 0 for the error term set ID, then the channel is currently using interpolated arrays. In this case, you can read from either the interpolated array (setID > 0) or the master array (setID = 0). **Note:**:Interpolated error terms are destroyed when no longer being used.
term	**(enum NAErrorTerm2)** - The error term to be retrieved. Choose from: 0 - naET_Directivity 1 - naET_SourceMatch 2 - naET_ReflectionTracking 3 - naET_TransmissionTracking 4 - naET_LoadMatch 5 - naET_Isolation
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
numPts	**(long integer)** - on input, max number of data points to return; on output: indicates the actual number of data points returned.
real()	**(single)** - array to accept the **real** part of the error-term. One-dimensional for the number of data points.
imag()	**(single)** - array to accept the **imaginary** part of the error-term. One-dimensional for the number of data points.

Return Type Single
Default Not Applicable

Examples

```
dim numpts as long
 numpts = ActiveChannel.NumberOfPoints
 ReDim r(numpts)  ' real part
 ReDim i(numpts)  ' imaginary part
 Dim CalSet as CalSet
 set CalSet = pna.GetCalManager.GetCal SetByGUID( txtGUID )
 Dim eData As ICalData2
 Set eData = CalSet
 eData.getErrorTermComplex 0, naET_LoadMatch, 1, 2, numpts,
r(0),i (0)
```

C++ Syntax HRESULT getErrorTermComplex(long setID, tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)

Interface ICalData2

Write-only **About Cal Sets**

PutErrorTermComplex Method

Description	Puts error term data into the Cal Set. Learn more about reading and writing Cal data using COM Before calling this method you must open the Cal Set with OpenCal Set. If the Cal Set is not open, this method returns E_NA_Cal Set_ACCESS_DENIED.
VB Syntax	*data.***putErrorTermComplex** *term, rcv, src, numPts, real(), imag()*

Variable	**(Type) - Description**
data	An ICalData2 pointer to the Cal Set object
term	**(enum NAErrorTerm2)** - The error term to be written. Choose from: 0 - naET_Directivity 1 - naET_SourceMatch 2 - naET_ReflectionTracking 3 - naET_TransmissionTracking 4 - naET_LoadMatch 5 - naET_Isolation
rcv	**(long integer)** - Receiver Port
src	**(long integer)** - Source Port
numPts	**(long integer) -** number of data points in the array
real()	**(single)** - array containing the **real** part of the calibration data. One-dimensional: the number of data points.
imag()	**(single)** - array containing the **imaginary** part of the calibration data. One-dimensional: the number of data points.

Return Type	Not Applicable
Default	Not Applicable
Examples	Dim eData As ICalData2 Set eData = app.GetCalManager.Cal Sets.Item(1) eData.putErrorTermComplex naET_LoadMatch, 1, 2, numpts, rel(0), img(0)
C++ Syntax	HRESULT putErrorTermComplex(tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)
Interface	ICalData2

Cal Set Object custom ICalData3 Interface

Read-only

GetErrorTermComplexByString

Description	Returns data for a specific error term.
VB Syntax	**GetErrorTermComplexByString**(*etermSetID, bufferName, lnumPoints, rea(0)l, imag(0)*)

Variable	**(Type) - Description**
bufferName	**(String)** The string used to identify a particular buffer in the calset. An example string for port 3 directivity in a full 2 port cal might be "Full 2 Port Cal (2,3)::Directivity(3,3)".

lnumpoints	**[In] (long)** The size of the float arrays in arguments"real" & "imag". **[Out] (long)** The number of actual data buckets returned in the real and imag arrays. see note
real	**(Single)** The real component of the complex data.
imag	**(Single)** The imaginary component of the complex data.

Return Type Single
Default Not Applicable

Examples See example

C++ Syntax **HRESULT PutErrorTermComplexByString(BSTR** bufferName**,**
long lnumPoints**,**
float* real**,**
float* imag**);**

Interface ICalData3

Note: *lnumpoints* is an In/Out parameter.
On the way **in**, it indicates the **max** number of values the user is requesting. This is critical because the capacity of the "read" & "imag" arrays must be allocated by the user to hold at least this number of elements. On the way **out,** it indicates the values actually returned.

Read-only

GetStandardComplexByString

Description Queries the calset for specific standard data.
VB Syntax **GetStandardComplexByString**(*bufferName, lnumPoints, rea(0)l, imag(0)*)

Variable	**(Type) - Description**
bufferName	(String) The string used to identify a particular buffer in the calset. An example string requesting the data for the Load standard in a full 2 port cal might be "Full 2 Port Cal (2,3)::S11C(3,3)".
lnumpoints	**[in] (Long)** The size of the arrays in arguments "real" & "imag". **[out] (Long)** The number of actual data buckets returned in the real and imag arrays. see note
real	**(Single)** The real component of the complex data.
imag	**(Single)** The imaginary component of the complex data.

Return Value Single
Default Not Applicable

Examples See example

C++ Syntax **HRESULT GetStandardComplexByString(BSTR** bufferName**,**
long* lnumPoints**,**
float* real**,**
float* imag**);**

Interface ICalData3

Note: *lnumpoints* is an In/Out parameter.

On the way **in**, it indicates the **max** number of values the user is requesting. This is critical because the capacity of the "read" & "imag" arrays must be allocated by the user to hold at least this number of elements. On the way **out,** it indicates the values actually returned.

Cal Sets Collection

Cal Sets Collection

Description

A collection object that provides a mechanism for iterating through all the Cal Sets in the analyzer. There is no ordering to the items in the collection. Therefore make no assumptions about the formatting of the collection. For more information, see Collections in the Analyzer.

Methods	**Description**
Item	Returns a handle to a Cal Set object in the collection.
Remove	Deletes the Cal Set residing at position index in the collection.
Properties	**Description**
Count	Returns the number of Cal Sets in the collection.

CalKit Object

CalKit Object **(default interface is ICalkit)**

Description

The calkit object provides the properties and methods to access and modify a calibration kit. The calkitType property can be set from either the **application object (app**.calKitType) or the **calKit object (calKit**.calKitType). Both of these commands specify or read the calibration kit type. When specified, the cal kit also becomes the Active cal kit. However, to retrieve a pointer to the cal kit, use **app.ActiveCalKit**.

The calKit object behaves somewhat differently from other objects in the system in that you can only have a pointer to **one** cal kit (which is also the active calkit).

Therefore, when you change the calkitType (from either of these objects) you may also be changing the object to which you may have several references. This is different from the behavior for most other objects in the system.

For example, the following code specifies two calKit type objects and in turn, assigns them to two different variables: ck1 and ck2.

```
Dim app As AgilentPNA835x.Application
 Dim ck1 As calKit
 Dim ck2 As calKit

 Private Sub Form_Load()
 Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")
 app.CalKitType = naCalKit_User1
 Set ck1 = app.ActiveCalKit

 app.CalKitType = naCalKit_User2
 Set ck2 = app.ActiveCalKit
```

```
Print "ck1: " & ck1.Name
Print "ck2: " & ck2.Name
End Sub
```

When the pointer to each of these kits is read (printed), they each have a pointer to the last kit to be assigned to the Active cal kit:

```
ck1: User Defined #1
ck2: User Defined #2
```

Method	**Description**
getCalStandard	Returns a handle to a calibration standard for modifying its definitions.
GetStandardsForClass	Returns the calibration standard numbers for a specified calibration class.
SetStandardsForClass	Sets the calibration standard numbers for a specified calibration class
Property	**Description**
CalKitType	Sets or returns the calibration kit type for to be used for calibration or for kit modification. Shared with the Application object.
Name	Sets and returns the name of the cal kit
PortLabel	Labels the ports for the kit; only affects the cal wizard annotation.
StandardForClass	**Obsolete** Maps a standard device to a cal class.

Write-only **About Modifying Cal Kits**

GetCalStandard Method

Description	Returns a handle to a calibration standard for modifying its definitions. To select a standard for performing a calibration (use Calibrator.AquireCalStandard).
VB Syntax	*calkit*.**GetCalStandard** *(index)*
Variable	**(Type) - Description**
calkit	A calKit **(object)**
index	**(long)** - Number of calibration standard. Choose **1** to **30;** (there are 30 cal standards in every kit).
Return Type	calStandard
Default	Not Applicable
Examples	Dim short As CalStandard Set short = calKit.getCalStandard(1) short.label = "myShort"
C++ Syntax	HRESULT GetCalStandard(long standardNumber, ICalStandard **pCalStd)
Interface	ICalKit

Read-only

GetStandardsForClass Method

Description	Get the calibration standard numbers for a specified calibration class. To set the calibration number use SetStandardsForClass Method
VB Syntax	*calkit*.**GetStandardsForClass** (*calclassorder, std1, std2, std3, std4, std5, std6, std7)*
Variable	**(Type) - Description**
calKit	A CalKit **(object)**
calclassorder	**(enum NACalClassOrder)** Choose from: 0 - naRefl_1_S11 1 - naRefl_2_S11 2 - naRefl_3_S11 3 - naTran_1_S21 4 - naRefl_1_S22 5 - naRefl_2_S22 6 - naRefl_3_S22 7 - naTran_1_S12 8 - naRefl_1_S33 9 - naRefl_2_S33 10 - naRefl_3_S33 11 - naTran_1_S32 12 - naTran_1_S23 13 - naTran_1_S31 14 - naTran_1_S13 15 - naTRL_T 16 - naTRL_R 17 - naTRL_L

std1…std7 — **(long)** Calibration Standard Number. Nominal values from **1** through **30**. **0** indicates that a standard number has not been selected.

Return Type — Not applicable
Default — Not applicable

Examples — calkit.GetStandardsForClass naRefl_3_S11, std1, std2, std3, std4, std5, std6, std7

C++ Syntax — HRESULT GetStandardsForClass(NACalClassOrder calclassorder, long std1, long std2, long std3, long std4, long std5, long std6, long std7)
Interface — ICalKit

Write/Read — **About Modifying Cal Kits**

Name (CalKit) Property

Description — Sets and Returns a name for the selected calibration kit.
VB Syntax — *calKit*.**Name** = *value*

Variable	(Type) - Description
calKit	A CalKit **(object).**
value	**(string)** -Calibration Kit name. Any string name, can include numerics, period, and spaces; any length (although the dialog box display is limited to about 30 characters).
Return Type	String
Default	Not Applicable

Examples

```
calKit.Name = "MyCalKit" 'Write
KitName = calKit.Name 'Read
```

C++ Syntax — HRESULT get_Name(BSTR *pVal)
HRESULT put_Name(BSTR newVal)
Interface — ICalKit

Write/Read — **About Modifying Cal Kits**

PortLabel Property

Description — Sets and returns the label on the calibration kit Port for the calibration wizard.
VB Syntax — *calKit*.**Portlabel** (*portNum*) = *value*

Variable	(Type) - Description
calKit	A CalKit **(object)**
portNum	**(long integer)** - number of the port to be labeled. Choose either **1** or **2**
value	**(string)** - Label that is visible in the calibration wizard.
Return Type	String
Default	Depends on the Cal Kit.

Examples	calKit.PortLabel = "MyCalKit" 'Write kitLabel = calKit.PortLabel 'Read
C++ Syntax	HRESULT get_PortLabel(long port, BSTR *pVal) HRESULT put_PortLabel(long port, BSTR newVal)
Interface	ICalKit

Write-only

SetStandardsForClass Method

Description	Set the calibration standard numbers for a specified calibration class. To read the cal standard numbers use GetStandardsForClass Method
VB Syntax	*calKit*.**SetStandardsForClass** *(calclassorder, std1, std2, std3, std4, std5, std6, std7)*

Variable	**(Type) - Description**
calKit	A CalKit **(object)**
calclassorder	**(enum NACalClassOrder)** Cal Class. Choose from: 0 - naRefl_1_S11 1 - naRefl_2_S11 2 - naRefl_3_S11 3 - naTran_1_S21 4 - naRefl_1_S22 5 - naRefl_2_S22 6 - naRefl_3_S22 7 - naTran_1_S12 8 - naRefl_1_S33 9 - naRefl_2_S33 10 - naRefl_3_S33 11 - naTran_1_S32 12 - naTran_1_S23 13 - naTran_1_S31

14 - naTran_1_S13

15 - naTRL_T

16 - naTRL_R

17 - naTRL_L

std1…std7	**(long)** Calibration Standard Number. Choose from **1** through **30**. Std2 through Std7 are optional
Return Type	Not applicable
Default	Not applicable
Examples	**calkit.SetStandardsForClass** naRefl_3_S11, 3, 5, 6 **calkit.SetStandardsForClass** naTran_1_S21, 4
C++ Syntax	HRESULT SetStandardsForClass(NACalClassOrder calclassorder, long std1, long std2, long std3, long std4, long std5, long std6, long std7)
Interface	ICalKit

Write/Read **About Modifying Cal Kits**

StandardForClass Property

Description	Sets a standard to a calibration class. Does NOT set or dictate the order for measuring the standards.
VB Syntax	*calKit.***StandardForClass***(class, portNum)* = *value*
Variable	**(Type) - Description**
calKit	A CalKit **(object).** Use calKit.GetCalStandard to get a handle to the standard.
class	**(enum NACalClass)** Standard. Choose from:

1 - naClassA

2 - naClassB

3 - naClassC

4 - naClassD

5 - naClassE

6 - naReferenceRatioLine

7 - naReferenceRatioThru

SOLT Standards

1 - naSOLT_Open

2 - naSOLT_Short

3 - naSOLT_Load

4 - naSOLT_Thru

5 - naSOLT_Isolation

TRL Standards

1 - naTRL_Reflection

2 - naTRL_Line_Reflection

3 - naTRL_Line_Tracking

4 - naTRL_Thru

5 - naTRL_Isolation

portNum **(long) -** The port number the standard will be connected to. For example, you may have a 3.5mm connector designated for port 1, and Type N designated for port 2.

value **(double)** - Calibration class number. Choose a number between **1** and **8.** The *<value>* numbers are associated with the following calibration classes:

<value>	**Class**	**Description**
1	S11A	Reflection standard
2	S11B	Reflection standard
3	S11C	Reflection standard
4	S21T	Thru standard
5	S22A	Reflection standard
6	S22B	Reflection standard
7	S22C	Reflection standard
8	S21T	Thru standard

Return Type
Default

Examples

C++ Syntax

Interface

CalManager Object

CalManager Object (default interface is ICalManager2)

Description

Use this interface to list, save, and delete Cal Sets.

ICalManager2 extends ICalManager with 3 Methods. This is the **default** interface.

Methods	**Interface**	**Description**
CreateCalSet	ICalManager	Creates a new Cal Set
CreateCustomCal	**ICalManager2**	Attempts to create a custom cal object.
DeleteCalSet	ICalManager	Deletes a Cal Set
GetCalSetByGUID	ICalManager	Get a handle to a Cal Set
GetCalSetCatalog	ICalManager	Gets a list of Cal Sets
GetCalSetUsageInfo	ICalManager	Returns the Cal Set ID and Error Term ID currently in use
GetCalTypes	**ICalManager2**	Query for a list of available calibration types.
GetRequiredEtermNames	**ICalManager2**	Returns an array of strings specifying the error terms required by the caltype's correction algorithm.
SaveCalSets	ICalManager	Writes new or changed Cal Sets to disk Shared with the Calibrator Object

Properties

CalSets (collection)

Write-only **About Cal Sets**

CreateCalSet Method

Description Creates a new Cal Set.

The new cal set is initialized with the stimulus settings from the channel whose number is passed as the argument to this method. Stimulus settings include frequency, bandwidth, number of points, etc.

Use this method when you want to manually upload data to the Cal Set using the returned ICal Set interface handle..

Note: The channel number does not restrict the usage of this Cal Set on any other channel. It simply provides a link to the originating channel so that the stimulus values can be stored in the Cal Set.

VB Syntax *calMgr.***CreateCalSet** *(chan)*

Variable	**(Type) - Description**
calMgr	**(object) -** A CalManager object
chan	**(long)** - channel number of the new Cal Set.
Return Type	ICal Set Interface
Default	Not Applicable

Example calMgr.CreateCalSet 1

C++ Syntax HRESULT CreateCalSet(long ChannelNumber, ICal Set** pCal Set);

Interface ICalManager

Write-only

CreateCustomCal

Description Attempts to create a custom cal object.

This method is exposed on the PNA's automation model to allow a remote DCOM user a convenient way of creating a custom calibration type. (Local users can create an in-process object).

VB Syntax **CreateCustomCal**("CalType.MyCalType")

Variable	**(Type) - Description**
CLSIDCustomCal :	[in] the CLSID or ProgID of the custom calibration type.
ppObject	[out] the IDispatch interface handle to the created object.
Return Type	String
Default	Not Applicable

Examples

VB example of using CreateCustomCal:

Set x = CreateCustomCal("CalType.MyCalType")

' in this example MyCalType is a bogus name

C++ Syntax HRESULT CreateCustomCal(BSTR CLSIDCustomCal, IDispatch** ppObject)

Interface ICalManager2

Write-only **About Cal Sets**

DeleteCalSet Method

Description Deletes a Cal Set from the set of available Cal Sets. This method immediately updates the Cal Set file on the hard drive. If the Cal Set is currently being used by a channel, this request will be denied and an error is returned.

Errors returned by this method:

E_NA_CAL_SET_IN_USE

E_NA_Cal Set_NOT_FOUND

E_NA_Cal Set_SAVE_FAILED

Using the Cal Sets collection is a convenient way to manage Cal Sets.

VB Syntax	*calMgr.***DeleteCalSet** *(GUID)*
Variable	**(Type) - Description**
calMgr	**(object) -** A CalManager object
GUID	**(string)** - GUID number of the Cal Set to be deleted
Return Type	Not Applicable
Default	Not Applicable

Example

```
dim cs As CalSet   ' the collection
dim strGUID as string


strGUID = cs.GetGUID
```

calMgr.DeleteCalSet strGUID

C++ Syntax	HRESULT DeleteCalSet(BSTR strGUID);
Interface	ICalManager

Read-only **About Cal Sets**

Get CalSetByGUID Method

Description	Requests a Cal Set by GUID. Returns an ICal Set interface.
VB Syntax	*calMgr.***GetCalSetByGUID** *(GUID)*
Variable	**(Type) - Description**
calMgr	**(object) -** A CalManager object
GUID	**(string)** - GUID of the Cal Set being requested.
Return Type	Interface object
Default	Not Applicable
Example	calMgr.GetCalSetByGUID (2B893E7A-971A-11d5-8D6C-00108334AE96)
C++ Syntax	HRESULT GetCalSetByGUID(BSTR* strGUID, ICal Set* pCalSet);
Interface	ICalManager

Read-only **About Cal Sets**

GetCalSetCatalog Method

Description

Returns a string containing a list of comma-separated GUIDs in the following format:

{FD6F863E-9719-11d5-8D6C-00108334AE96},

	{1B03B2CE-971A-11d5-8D6C-00108334AE96},
	{2B893E7A-971A-11d5-8D6C-00108334AE96}
VB Syntax	*value = calMgr.*__GetCalSetCatalog__

Variable	**(Type) - Description**
value	**(string)** - Variable to store the returned GUID list
calMgr	**(object) -** A CalManager object
Return Type	String
Default	Not Applicable

Example	value = calMgr.GetCalSetCatalog

C++ Syntax	HRESULT GetCalSetCatalog(BSTR);
Interface	ICalManager

Read-only **About Cal Sets**

GetCalSetUsageInfo Method

Description

Returns a string identifying the Cal Set currently in use by the specified channel.

This method identifies the Cal Set being used by returning its GUID.

This method also identifies the "Error Term set" within the Cal Set.

Error term sets are identified by integers, with set 0 belonging to the original (non-interpolated) terms. As stimulus values for a channel are changed causing interpolation to be required, a new Error Term set is constructed within the Cal Set to hold the interpolated Error Terms. The sets are sequentially numbered 1, 2, 3, and so forth. These Error Term sets are destroyed when they are no longer being used.

If there is no Cal Set in use for the given channel, the <GUID> argument is set to the empty string.

VB Syntax	*calMgr.*__GetCalSetUsageInfo__ (*chan, GUID, EtermID*)

Variable	**(Type) - Description**
calMgr	**(object) -** A CalManager object
chan	**(long [in]) -** channel of the Cal Set being requested
GUID	**(string [out])** - variable to store the GUID of the Cal Set being requested. If there is no Cal Set in use for the given channel, the <GUID> argument is set to the empty string.
EtermID	**(long [out]) -** variable to store the error term ID being requested. If the returned argument is greater than 0, the set is being interpolated.
Return Type	String , Long Integer
Default	Not Applicable

Example	calMgr.GetCalSetUsageInfo (1, GUID, EtermID)

C++ Syntax	HRESULT GetCalSetUsageInfo (long lChannel, BSTR* CalSetGUID, long* etermSetID);
Interface	ICalManager

Read

GetCalTypes

Description	Query for a list of available calibration types. It returns a two dimensional array of strings identifying all the available calibration types.(see note)
VB Syntax	v = pna.GetCalManager.**GetCalTypes**
Variable	**(Type) - Description**
v	NameGuidPair: [out] contains the calibration type name and associated GUID for each cal type known to the PNA
Return Type	This method returns a **variant** of type VT_ARRAY. The array is two dimensional containing elements of type variant.
Default	Not Applicable

Examples

```
Dim pna
Dim v
Dim i
    Set pna = CreateObject("AgilentPNA835x.Application")
v = pna.GetCalManager.GetCalTypes
Dim str          For i = LBound(v, 1) To UBound(v, 1)
              str = str & vbcrlf  & v(i, 0) & vbtab & v(i, 1)
           Next
MsgBox str
```

C++ Syntax	HRESULT GetCalTypes(VARIANT * NameGuidPair)
Interface	ICalManager2

Note: This method returns "all" CalTypes both Standard and Custom. The Standard CalTypes are the same on all PNA's, but the Custom CalTypes are not necessarily the same. They are dependent on what Custom components have been installed in the instrument.

Write-only

GetRequiredEtermNames

Description	Returns an array of strings specifying the error terms required by the caltype's correction algorithm in order to correct the specified parameter. This function interrogates a specific caltype (caltypeGUID) for the list of error terms it would need in order to correct the specified parameter. All the standard S Parameter calibration types embed port specifiers in the error term name. The specific port information is gleaned from the passed parameter. For example, to query the error term requirements specific to a two port cal on ports 1 and 3, issue this with a parameter of S13 or S31. The buffer names returned will be formatted in this way: Full 1 Port SOLT(1,3):TransmissionTracking(3,1)
VB Syntax	**EtermNames = GetRequiredEtermNames**(CalTypeGUID As String, Parameter As String)
Variable	**(Type) - Description**
caltypeGUID:	[in] the GUID of the desired calibration type
parameter	[in] string specifying the parameter to be corrected
EtermNames	[out] array of strings containing the error term names.

	Note: In C++ Allocated by server. Must be freed by caller using SysFreeString.
Return Type	Not Applicable
Default	Not Applicable
Examples	enames = GetRequiredEtermNames(ctGUID, Parm)
C++ Syntax	HRESULT GetRequiredEtermNames(BSTR caltypeGUID, BSTR parameter, VARIANT* EtermNames)
Interface	ICalManager2

CalStandard Object

CalStandard Object (default interface is ICalStandard2)

Description

Contains all of the settings that are required to modify a calibration kit. Get a handle to a standard with the calkit.GetCalStandard Method.

The **ICalStandard2** interface extends the CalStandard interface by allowing setting and reading complex impedance values. Use this as the **default interface**.

Method

None

Property	**Interface**	**Description**
C0	ICalStandard	Sets and Returns the C0 (C-zero) value (the first capacitance value) for the calibration standard, when the Type is set to "naOpen".
C1	ICalStandard	Sets and Returns the C1 value (the second capacitance value) for the calibration standard, when the Type is set to "naOpen".
C2	ICalStandard	Sets and Returns the C2 value (the third capacitance value) for the calibration standard, when the Type is set to "naOpen".
C3	ICalStandard	Sets and Returns the C3 value (the fourth capacitance value) for the calibration standard, when the Type is set to "naOpen".
Delay	ICalStandard	Sets and Returns the electrical delay value for the calibration standard.
L0	ICalStandard	Sets and Returns the L0 (L-zero) value (the first inductance value) for the calibration standard, when the Type is set to "naShort".
L1	ICalStandard	Sets and Returns the L1 value (the second inductance value) for the calibration standard, when the Type is set to "naShort"..
L2	ICalStandard	Sets and Returns the L2 value (the third inductance value) for the calibration standard, when the Type is set to "naShort"..
L3	ICalStandard	Sets and Returns the L3 value (the third inductance value) for the calibration standard, when the Type is set to "naShort"..
Label	ICalStandard	Sets and Returns the label for the calibration standard.
loss	ICalStandard	Sets and Returns the insertion loss for the calibration standard.
Maximum Frequency	ICalStandard	Sets and Returns the maximum frequency for the calibration standard.

Medium	ICalStandard	Sets and Returns the media type of the calibration standard.
Minimum Frequency	ICalStandard	Sets and Returns the minumum frequency for the calibration standard.
Type	ICalStandard	Sets and Returns the type of calibration standard. Selections are: **naOpen, naShort, naLoad, naThru, naArbitraryImpedance and naSliding.**
TZReal	**ICalStandard2**	Sets and Returns the TZReal value (the Real Terminal Impedance value) for the calibration standard, when the Type is set to "naArbitraryImpedance".
TZImag	**ICalStandard2**	Sets and Returns the TZImag value (the Imaginary Terminal Impedance value) for the calibration standard, when the Type is set to "naArbitraryImpedance".
Z0	ICalStandard	Sets and Returns the characteristic impedance for the calibration standard.

Write/Read **About Modifying Cal Kits**

C0 Property

Description Sets and Returns the C0 (C-zero) value (the first capacitance value) for the calibration standard.

To set the other capacitance values, use C1, C2, C3

VB Syntax *calstd*.**C0** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for C0 in picofarads
Return Type	Single
Default	Not Applicable

Examples

```
calstd.C0 = 15 'Write the value of C0 to 15picofarads
cap0 = calstd.C0 'Read the value of C0
```

C++ Syntax

```
HRESULT get_C0(float *pVal)
HRESULT put_C0(float newVal)
```

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

C1 Property

Description Sets and Returns the C1 value (the second capacitance value) for the calibration standard.

To set the other capacitance values, use C0, C2, C3

VB Syntax *calstd*.**C1** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for C1 in picofarads
Return Type	Single
Default	Not Applicable

Examples

calstd.C1 = 15 'Write the value of C1 to 15picofarads
cap1 = calstd.C1 'Read the value of C1

C++ Syntax

HRESULT get_C1(float *pVal)
HRESULT put_C1(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

C2 Property

Description

Sets and Returns the C2 value (the third capacitance value) for the calibration standard.

To set the other capacitance values, use C0, C1, C3

VB Syntax *calstd*.**C2** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for C2 in picofarads
Return Type	Single
Default	Not Applicable

Examples

calstd.C2 = 15 'Write the value of C2 to 15picofarads
cap2 = calstd.C2 'Read the value of C2

C++ Syntax

HRESULT get_C2(float *pVal)
HRESULT put_C2(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

C3 Property

Description

Sets and Returns the C3 value (the fourth capacitance value) for the calibration standard.

To set the other capacitance values, use C0, C1, C2

VB Syntax *calstd*.**C3** = *value*

Variable	(Type) - Description
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for C3 in picofarads
Return Type	Single
Default	Not Applicable

Examples

calstd.C3 = 15 'Write the value of C3 to 15picofarads
cap3 = calstd.C3 'Read the value of C3

C++ Syntax

HRESULT get_C3(float *pVal)
HRESULT put_C3(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

Delay Property

Description Sets and Returns the electrical delay value for the calibration standard.

VB Syntax *calstd*.**Delay** = *value*

Variable	(Type) - Description
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Electrical delay in seconds
Return Type	Single
Default	Not Applicable

Exaamples

calstd.Delay = .00015 'Write the Delay .00015 seconds
stdDelay = calstd.Delay 'Read the value of Delay

C++ Syntax

HRESULT get_Delay(float *pVal)
HRESULT put_Delay(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

L1 Property

Description Sets and Returns the L1 value (the second inductance value) for the calibration standard.

To set the other inductance values, use L0, L2, L3

VB Syntax *calstd*.**L1** = *value*

Variable	(Type) - Description
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for L1 in picohenries
Return Type	Single

Default Not Applicable

Examples

calstd.L1 = 15 'Write the value of L1 = 15picohenries
Induct1 = calstd.L1 'Read the value of L1

C++ Syntax

HRESULT get_L1(float *pVal)
HRESULT put_L1(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

L2 Property

Description

Sets and Returns the L2 value (the third inductance value) for the calibration standard.

To set the other inductance values, use L0, L1, L3

VB Syntax *calstd*.**L2** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for L2 in picohenries

Return Type Single

Default Not Applicable

Examples

calstd.L2 = 15 'Write the value of L2 to 15picohenries
Induct2 = calstd.L2 'Read the value of L2

C++ Syntax

HRESULT get_L2(float *pVal)
HRESULT put_L2(float newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

L3 Property

Description

Sets and Returns the L3 value (the third inductance value) for the calibration standard.

To set the other inductance values, use L0, L1, L2

VB Syntax *calstd*.**L3** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for L3 in picohenries

Return Type Single

Default Not Applicable

Examples

calstd.L3 = 15 'Write the value of L3 to 15picohenries
Induct3 = calstd.L3 'Read the value of L3

C++ Syntax	HRESULT get_L3(float *pVal) HRESULT put_L3(float newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

L0 Property

Description	Sets and Returns the L0 (L-zero) value (the first inductance value) for the calibration standard. To set the other inductance values, use L1, L2, L3
VB Syntax	*calstd*.**L0** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for L0 in picohenries
Return Type	Single
Default	Not Applicable
Examples	calstd.L0 = 15 'Write the value of L0 = 15picohenries Induct0 = calstd.L0 'Read the value of L0
C++ Syntax	HRESULT get_L0(float *pVal) HRESULT put_L0(float newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

Label Property

Description	Sets and Returns the label for the calibration standard. The label is used to prompt the user to connect the specified standard.
VB Syntax	*calstd*.**Label** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(string)** - between 1 and 12 characters long. Cannot begin with a numeric.
Return Type	String
Default	Not Applicable
Examples	calstd.Label = "Short" 'Write stdLabel = calstd.Label 'Read
C++ Syntax	HRESULT get_Label(BSTR *pVal) HRESULT put_Label(BSTR newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

Loss Property

Description	Sets and Returns the insertion loss for the calibration standard.
VB Syntax	*calstd*.**loss** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Insertion loss in Mohms / sec. (MegaOhms per second of electrical delay)
Return Type	Single
Default	Not Applicable
Examples	calstd.loss = 3.5e9 'Write stdLoss = calstd.loss 'Read the value of Loss
C++ Syntax	HRESULT get_Loss(float *pVal) HRESULT put_Loss(float newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

MaximumFrequency Property

Description	Sets and Returns the maximum frequency for the calibration standard.
VB Syntax	*calstd*.**MaximumFrequency** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(double)** - Maximum frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	calstd.MaximumFrequency = 9e9 'Write maxFrequency = calstd.MaximumFrequency 'Read
C++ Syntax	HRESULT get_MaximumFrequency(double *pVal) HRESULT put_MaximumFrequency(double newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

Medium Property

Description	Sets and Returns the media type of the calibration standard.
VB Syntax	*calstd*.**Medium** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(enum** NACalStandardMedium**)** - Medium of the transmission line of the standard. Choose from: **0 - naCoax** - Coaxial Cable **1 - naWaveGuide**
Return Type	Long Integer
Default	Not Applicable
Examples	calstd.Medium = naCoax 'Write stdMedium = calstd.Medium 'Read
C++ Syntax	HRESULT get_Medium(tagNACalStandardMedium *pVal) HRESULT put_Medium(tagNACalStandardMedium newVal)
Interface	ICalStandard

Write/Read **About Modifying Cal Kits**

MinimumFrequency Property

Description	Sets and Returns the minimum frequency for the calibration standard.
VB Syntax	*calstd*.**MinimumFrequency** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(double)** -Minimum frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	calstd.MinimumFrequency = 300e3 'Write minFrequency = calstd.MinimumFrequency 'Read
C++ Syntax	HRESULT get_MinimumFrequency(double *pVal) HRESULT put_MinimumFrequency(double newVal)
Interface	ICalStandard

Write/Read. **About Modifying Cal Kits**

Type (calstd) Property

Description	Sets and Returns the type of calibration standard.
VB Syntax	*calstd*.**Type** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the

	standard.
value	**(enum** NACalStandardType**)** -Choose from: **0 - naOpen** **1 - naShort** **2 - naLoad** **3 - naThru**
Return Type	Long Integer
Default	Not Applicable

Examples

calstd.Type = naOpen 'Write
standardtype = calstd.Type 'Read

C++ Syntax

HRESULT get_Type(tagNACalStandardType *pVal)
HRESULT put_Type(tagNACalStandardType newVal)

Interface ICalStandard

Write/Read **About Modifying Cal Kits**

TZImag Property

Description

Sets and Returns the TZImag value (the Imaginary Terminal Impedance value) for the calibration standard. Only applicable when "Type" is set to **naArbitraryImpedance.**

To set the other resistance values, use TZReal

VB Syntax *calstd*.**TZImag** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for TZImag in Ohms
Return Type	Single
Default	Not Applicable

Examples

calstd.TZImag = 15 'Write the value of TZImag to 15 Ohms
imp0 = calstd.TZImag 'Read the value of TZImag

C++ Syntax

HRESULT TZImag([out, retval] float *pVal);
HRESULT TZImag([in] float newVal);

Interface ICalStandard2

Write/Read **About Modifying Cal Kits**

TZReal Property

Description

Sets and Returns the TZReal value (the real Terminal Impedance value) for the calibration standard. Only applicable when "Type" is set to **naArbitraryImpedance.**

To set the other resistance values, use TZImag

VB Syntax *calstd*.**TZReal** = *value*

Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** - Value for TZReal in Ohms
Return Type	Single
Default	Not Applicable
Examples	calstd.TZReal = 15 'Write the value of TZReal to 15 Ohms imp0 = calstd.TZReal 'Read the value of TZReal
C++ Syntax	HRESULT TZReal([out, retval] float *pVal); HRESULT TZReal([in] float newVal);
Interface	ICalStandard2

Write/Read **About Modifying Cal Kits**

Z0 Property

Description	Sets and Returns the characteristic impedance for the calibration standard.
VB Syntax	*calstd*.**Z0** = *value*
Variable	**(Type) - Description**
calstd	A CalStandard **(object).** Use calKit.GetCalStandard to get a handle to the standard.
value	**(single)** -Impedance in Ohms
Return Type	Single
Default	Not Applicable
Examples	calstd.Z0 = 50 'Write impedance = calstd.Z0 'Read
C++ Syntax	HRESULT get_Z0(float *pVal) HRESULT put_Z0(float newVal)
Interface	ICalStandard

Channel Object

Channel Object (default interface is IChannel2)

Description

The channel object is like the engine that produces data. Channel settings consist of stimulus values like frequency, power, IF bandwidth, and number of points.

You can get a handle to a channel in a number of ways. But first you have to make sure that the channel exists. When you first startup the analyzer, there is one S11 measurement on channel 1. Thus there is only one channel in existence. You can do the following:

```
Dim chan as Channel
```

```
'
Set chan = pna.ActiveChannel
```

or

```
Set chan = pna.Channels( n )
```

The first method will return the channel object that is driving the active measurement. When you ask for the ActiveChannel, you get the channel that is driving the active measurement. If there is no measurement, there may not be a channel. Once a channel is created, it does not go away. So if there once was a measurement (hence a channel), the channel will still be available. If there is no channel you can create one in a couple ways. Here's one way:

```
Pna.CreateMeasurement( ch1, "S11", port1, window2)
```

Here's another:

```
Pna.Channels.Add (ch2)
```

The latter will have no visible effect on the analyzer. It will simply create channel 2 if it does not already exist.

The **IChannel2 interface** extends the Channel interface. It provides frequency offset capability required for measuring frequency converting devices. Ichannel2 is the **default** interface

Also see **custom** ISourcePowerCalData Interface

Method	**Interface**	**Description**
Abort	IChannel	Aborts the current measurement sweep on the channel.
AveragingRestart	IChannel	Clears and restarts averaging of the measurement data.
Continuous	IChannel	The channel continuously responds to trigger signals.
CopyToChannel	IChannel	Sets up another channel as a copy of this object's channel.
getSourcePowerCalData	IChannel	Returns requested source power calibration data, if it exists.
GetXAxisValues	IChannel	Returns the channel's X-axis values into a dimensioned Variant array.
GetXAxisValues2	IChannel	Returns the channel's X-axis values into a dimensioned NON-Variant array.
Hold	IChannel	Puts the Channel in Hold - not sweeping.
Next_IFBandwidth	IChannel	A function that returns the Next higher IF Bandwidth value.
NumberOfGroups	IChannel	Sets the Number of trigger signals the channel will receive.
Preset	IChannel	Resets the channel to factory defined settings.
PreviousIFBandwidth	IChannel	Returns the previous IF Bandwidth value.
putSourcePowerCalData	IChannel	Inputs source power calibration data to this channel for a specific source port.
SelectCal Set	IChannel	Specifies the Cal Set to use for the Channel
Single	IChannel	Channel responds to one trigger signal from any source (internal, external, or manual). Then channel switches to Hold.
Property	**Interface**	**Description**
AlternateSweep	IChannel	Sets sweeps to either alternate or chopped.
Attenuator	IChannel	Sets or returns the value of the attenuator control for the specified port number.

AttenuatorMode	IChannel	Sets or returns the mode of operation of the attenuator control for the specified port number.
Averaging	IChannel	Turns trace averaging ON or OFF for all measurements on the channel.
AveragingCount	IChannel	Returns the number of sweeps that have been averaged into the measurements.
AveragingFactor	IChannel	Specifies the number of measurement sweeps to combine for an average.
Calibrator (object)		
centerFrequency	IChannel	Sets or returns the center frequency of the channel. Shared with the Segment Object
channelNumber	IChannel	Returns the Channel number. Shared with the Measurement Object
CouplePorts	IChannel	Turns ON and OFF port power coupling.
CWFrequency	IChannel	Set the Continuous Wave (CW) frequency.
DwellTime	IChannel	Sets or returns the dwell time for the channel. Shared with the Segment Object
FrequencyOffsetDivisor	**IChannel 2**	Part of formula used to determine offset frequency of receivers
FrequencyOffsetFrequency	**IChannel 2**	Part of formula used to determine offset frequency of receivers
FrequencyOffsetMultiplier	**IChannel 2**	Part of formula used to determine offset frequency of receivers
FrequencyOffsetCWOverride	**IChannel 2**	Establishes a fixed (CW) stimulus frequency while measuring swept response frequency range.
FrequencyOffsetState	**IChannel 2**	Turns frequency Offset ON and OFF
FrequencySpan	IChannel	Sets or returns the frequency span of the channel. Shared with the Segment Object
IFBandwidth	IChannel	Sets or returns the IF Bandwidth of the channel. Shared with the Segment Object
NumberOfPoints	IChannel	Sets or returns the Number of Points of the channel. Shared with the Segment Object
Parent	IChannel	Returns a handle to the parent object of the channel.
PowerSlope	IChannel	Sets or returns the Power Slope value.
R1InputPath	**IChannel 2**	Throws internal reference switch (option 081)
ReceiverAttenuator	IChannel	Sets or returns the value of the specified receiver attenuator control.
Segments (collection)		
SourcePowerCorrection	IChannel	Turns source power correction ON or OFF for a specific source port.
StartFrequency	IChannel	Sets or returns the start frequency of the channel. Shared with the Segment Object
StartPower	IChannel	Sets the start power of the analyzer when sweep type is set to Power Sweep.
StopFrequency	IChannel	Sets or returns the stop frequency of the channel. Shared with the Segment Object
StopPower	IChannel	Sets the Stop Power of the analyzer when sweep type is set to Power Sweep.
SweepGenerationMode	IChannel	Sets the method used to generate a sweep: continuous ramp (analog) or discrete steps (stepped).

SweepTime	IChannel	Sets the Sweep time of the analyzer.
SweepType	IChannel	Sets the type of X-axis sweep that is performed on a channel.
TestPortPower	IChannel	Sets or returns the RF power level for the channel. Shared with the Segment Object
TriggerMode	IChannel	Determines the measurement that occurs when a trigger signal is sent to the channel.
UserRangeMax	IChannel	Sets the stimulus stop value for the specified User Range.
UserRangeMin	IChannel	Sets the stimulus start value for the specified User Range.
XAxisPointSpacing	IChannel	Sets X-Axis point spacing for the active channel.

ISourcePowerCalData Interface

Description

Contains methods for putting source power calibration data in and getting source power calibration data out of the analyzer using typed data. The methods in this interface transfer data more efficiently than methods that use variant data.

Method	**Description**
getSourcePowerCalDataScalar	Returns requested source power calibration data, if it exists.
putSourcePowerCalDataScalar	Inputs source power calibration data to a channel, for a specific source port.
Property	**Description**
None	

Write-only **About Triggering**

Abort Method

Description	Ends the current measurement sweep on the channel.
VB Syntax	*chan*.**Abort** [*sync*]
Variable	**(Type) - Description**
chan	**(object)** - A Channel object
sync	**(boolean)** - wait (or not) for the analyzer to stop before processing subsequent commands. Optional argument; if unspecified, value is set to False. Choose from: **True** - synchronize - the analyzer will not process subsequent commands until the current measurement is aborted. **False** - continue processing commands immediately
Return Type	None
Default	None
Examples	chan.abort True chan.abort
C++ Syntax	HRESULT Abort(VARIANT_BOOL bSynchronize);

Interface IChannel

Write-only **About Averaging**

AveragingRestart Method

Description	Clears and restarts averaging of the measurement data.
VB Syntax	*chan*.**AveragingRestart**
Variable	**(Type) - Description**
chan	A Channel **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	chan.AveragingRestart
C++ Syntax	HRESULT AveragingRestart()
Interface	IChannel

Write-only **About Triggering**

Continuous Method

Description	The channel continuously responds to trigger signals. **Note:** This command does **NOT** change TriggerSignal to Continuous.
VB Syntax	*chan*.**Continuous**
Variable	**(Type) - Description**
chan	A Channel **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	chan.Continuous
C++ Syntax	HRESULT Continuous()
Interface	IChannel

Write-only

CopyToChannel Method

Description	Sets up another channel as a copy of this object's channel.
VB Syntax	chan.**CopyToChannel**(lChanNum)
Variable	**(Type) - Description**
chan	A Channel **(object)**
lChanNum	**(long integer)** – Number of the channel to become a copy of this channel.
Return Type	None
Default	Not Applicable

Examples Dim chan As Channel

Set chan = PNAapp.ActiveChannel

Const lNEW_CHAN_NUM As Long = 2
chan.CopyToChannel(lNEW_CHAN_NUM)

C++ Syntax HRESULT CopyToChannel(long lChanNum);
Interface IChannel2

Read-only **About Source Power Cal**

getSourcePowerCalData Method

Description Retrieves (as variant data type) requested source power calibration data, if it exists, from this channel.

Note: This method returns a variant which is less efficient than methods available on the ISourcePowerCalData interface

VB Syntax *data* = *chan*.**getSourcePowerCalData** *sourcePort*

Variable	**(Type) - Description**
data	**(variant)** – Array to store the data.
chan	**(object)** – A Channel object
sourcePort	**(long integer)** – The source port for which calibration data is being requested.
Return Type	Variant array – automatically dimensioned to the size of the data.
Default	Not Applicable

Examples

```
Dim varData As Variant
 Const port1 As Long = 1
 varData = chan.getSourcePowerCalData port1
 'Print the data
 For i = 0 to chan.NumberOfPoints - 1
 Print varData(i)
 Next i
```

C++ Syntax HRESULT getSourcePowerCalData(long sourcePort, VARIANT *pData);
Interface IChannel

Read-only **About Segment Sweep**

GetXAxisValues2 Method

Description Returns the channel's X-axis values into a dimensioned Typed array. GetXAxisValues2 is a convenient method for determining the frequency of each point when the points are not linearly spaced - as in segment sweep.

Note: This method will fail if called using a scripting client such as VBScript or Agilent Vee, (see remarks)

Note: In Segment Sweep, chan.NumberofPoints will return the total number of data points for the combined segments.

VB Syntax *chan*.**GetXAxisValues2** *numPts,data*

Variable	**(Type) - Description**
chan	**(object)** - A Channel object
numPts	**(long integer)** - Number of data points in the channel
data	**(double)** Single dimensioned array of data matching the number of points in the channel.
Return Type	double
Default	Not applicable

Examples

```
Dim App As Application
 Set App = New Application
 Dim numPoints As Long
 Dim values() As Double
 numPoints = App.ActiveChannel.NumberOfPoints
 ReDim values(numPoints)
 App.ActiveChannel.GetXAxisValues2 numPoints, values(0)
 Print values(0), values(1)
```

C++ Syntax	HRESULT GetXAxisValues2(long* pNumValues, double* stimulus)
Interface	IChannel

Remarks:

This method will fail if called using a scripting client such as VBScript or Agilent Vee.

This method also cannot be called using late-bound typing in Visual Basic. For instance, if, in the example above, the first line were replaced with "Dim App as Object", then this method would fail.

Use the GetXAxisValues method as a replacement. This method works for these COM environments.

Read-only **About Segment Sweep**

GetXAxisValues Method

Description	Returns the channel's X-axis values. GetXAxisValues is a convenient method for determining the frequency of each point when the points are not linearly spaced - as in segment sweep. **Note:** This method returns a variant which is less efficient than GetXAxisValues2. **Note**: In Segment Sweep, chan.NumberofPoints will return the total number of data points for the combined segments.
VB Syntax	*data = chan.***GetXAxisValues**
Variable	**(Type) - Description**
data	Variant array to store the data.
chan	A Channel **(object)**
Return Type	Variant
Default	Not Applicable

Examples

```
Dim varData As Variant
```

```
Dim i As Integer
varData = chan.GetXAxisValues
'Print Data
For i = 0 To chan.NumberOfPoints - 1
 Print varData(i)
Next i
```

C++ Syntax	HRESULT GetXAxisValues (VARIANT* xData)
Interface	IChannel

Write-only **About Triggering**

Hold Method

Description	Puts the Channel in Hold - not sweeping.
VB Syntax	*chan.***Hold** *[sync]*
Variable	**(Type) - Description**
chan	A Channel **(object)**
[sync]	**(boolean)** - Optional argument. A variable set to either True or False. **True** - program control waits until the channel is in the Hold state. **False** - program control continues immediately. You are not guaranteed the channel is in Hold yet.
Return Type	Not Applicable
Default	Not Applicable
Examples	wate = True chan.Hold wate
C++ Syntax	HRESULT Hold(VARIANT_BOOL bWait)
Interface	IChannel

Write-only **About Dynamic Range**

NextIFBandwidth Method

Description	A function that returns the Next higher IF Bandwidth value. Use to retrieve the list of available IFBandwidth settings.
VB Syntax	*chan.***Next_IFBandwidth** *bw*
Variable	**(Type) - Description**
chan	A Channel **(object)**
bw	**(double)** - The argument that you use to send an IFBandwidth. The function uses this argument to return the Next higher IFbandwidth.
Return Type	Double
Default	Not Applicable
Examples	**Public pnbw As Double 'declare variable outside of procedure** pnBW = chan.IFBandwidth 'put the current IFBW in pnBW chan.Next_IFBandwidth pnBW 'function returns the Next higher IFBandwidth.

chan.IFBandwidth = pnBW 'set IFBW to the Next value

C++ Syntax HRESULT Next_IFBandwidth (double *pVal)
Interface IChannel

Write-only **About Triggering**

NumberOfGroups Method

Description Sets the Number of trigger signals the channel will receive. After the channels has received that number of trigger signals, the channel switches to Hold mode.

To begin sweeping the number of groups, send app.Continuous

VB Syntax *chan*.**NumberOfGroups** *num, sync*

Variable	(Type) - Description
chan	A Channel **(object)**
num	**(long integer)** Number of trigger signals the channel will receive. Choose any number between 1 and 2 million
sync	**(boolean)** Variable set to either: **True -** subsequent commands are not processed until the groups are complete. **Do not use with manual trigger.** **False -** subsequent commands are processed immediately
Return Type	Not Applicable
Default	Not Applicable

Examples chan.NumberOfGroups

C++ Syntax HRESULT NumberOfGroups(long count, VARIANT_BOOL bWait)
Interface IChannel

Write-only **About Dynamic Range**

PreviousIFBandwidth Method

Description A function that returns the previous IF Bandwidth value. Use to retrieve the list of available IFBandwidth settings.

VB Syntax *chan.***Previous_IFBandwidth** *bw*

Variable	(Type) - Description
chan	A Channel **(object)**
bw	**(double)** - The argument that you use to send an IFBandwidth. The function uses this argument to return the previous IFbandwidth.
Return Type	Double
Default	Not Applicable

Examples

```
Public pnbw As Double 'declare variable outside of procedure
```

PreBW = chan.IFBandwidth 'put the current IFBW in PreBW
chan.Previous_IFBandwidth PreBW 'function returns the Previous IFBandwidth of the current one.
chan.IFBandwidth = PreBW 'set IFBW to the previous value

C++ Syntax	HRESULT Previous_IFBandwidth (double *pVal)
Interface	IChannel

Write-only **About Source Power Cal**

putSourcePowerCalData Method

Description	Inputs source power calibration data (as variant data type) to this channel for a specific source port.
VB Syntax	*chan*.**getSourcePowerCalData** *sourcePort, data*

Variable	**(Type) - Description**
chan	**(object)** – A Channel object
sourcePort	**(long integer)** – The source port for which calibration data is being requested.
data	**(variant)** – Array of source power cal data being input.
Return Type	None
Default	Not Applicable

Examples	chan.putSourcePowerCalData 1, varData

C++ Syntax	HRESULT putSourcePowerCalData(long sourcePort, VARIANT varData);
Interface	IChannel

Write-only

SelectCalSet Method

Description

Selects a Cal Set to apply to the measurements on the calling channel.

If the cal set's GUID is not found, this method returns E_NA_Cal Set_NOT_FOUND.

Note: Error Correction is not automatically applied as a result of this command being issued. If there is more than one Cal Type in the Cal Set, you must explicitly choose the Cal Type you want to apply. (See meas.Caltype)l

VB Syntax *channel.***SelectCalSet** *GUID, restore*

Variable **(Type) - Description**

channel **(object) -** A Channel object

GUID **(string)** - GUID number of the Cal Set to select

restore **(boolean)** -

True (1) - The stimulus stored with the cal set will be applied to the channel.

False (0) - If a conflict is detected between the existing channel settings and the Cal Set stimulus settings, then the following will occur:

If interpolation is ON, then interpolation will be attempted. This may fail if the channel frequency is outside the range of the Cal Set.

If interpolation is OFF, the selection will be abandoned and an error is returned: E_NA_CAL_STIMULUS_VALUES_EXCEEDED

Return Type	Not Applicable
Default	Not Applicable
Example	channel.SelectCalSet GUID, 1
C++ Syntax	HRESULT SelectCalSet (BSTR strGUID, bool bRestore);
Interface	IChannel

Write-only **About Triggering**

Single Method

Description	Sets the trigger count to 1, which will cause the channel to respond to exactly one trigger signal from any source (internal, external, or manual).
VB Syntax	*chan*.**Single** *[sync]*
Variable	**(Type) - Description**
chan	A Channel **(object)**
[sync]	**(boolean)** -Optional argument. A variable set to either True or False. **True** - The analyzer waits until the trigger is completed to process subsequent commands. **False** - Subsequent commands are processed immediately.
Return Type	Not Applicable
Default	Not Applicable
Examples	sync = True chan.Single sync
C++ Syntax	HRESULT Single(VARIANT_BOOL bWait)
Interface	IChannel

Write/Read **About Sweeping**

AlternateSweep Property

Description	Sets sweeps to either alternate or chopped.
VB Syntax	*chan*.**AlternateSweep** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(boolean)** - Choose either: **False (0)** - Sweep mode set to **Chopped** - reflection and transmission are measured on the same sweep. **True (1)** - Sweep mode set to **Alternate** - reflection and transmission measured on separate sweeps. Improves Mixer bounce and Isolation measurements. Increases cycle time.
Return Type	boolean
Default	False (0)
Examples	chan.AlternateSweep = True 'Write

altSwp = chan.AlternateSweep 'Read

C++ Syntax	HRESULT AlternateSweep(VARIANT_BOOL *pVal) HRESULT AlternateSweep(VARIANT_BOOL newVal)
Interface	IChannel

Read-only

Application Property

Description	Returns the name of the Analyzer making measurements on the channel.
VB Syntax	*chan.***Application**
Variable	**(Type) - Description**
chan	A Channel **(object)**
Return Type	object
Default	None
Examples	rfna = chan.Application 'returns the Analyzer name
C++ Syntax	HRESULT get_Application(IApplication** Application)
Interface	IChannel

Write/Read **About Attenuation**

AttenuatorMode Property

Description	Sets or returns the mode of operation of the attenuator control for the specified port number. This command is automatically set to Manual when an Attenuator value is set.
VB Syntax	*chan.***AttenuatorMode(***portNum***)** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
portNum	**(long)** - Port number (**1** or **2**) of attenuator control to be changed.
value	**(enum NAModes)** - Choose from: **0 - naAuto** - Attenuator control set to automatic. The analyzer will set the attenuator control appropriately to deliver the specified power at the source. **1 - naManual -** Specify the attenuator setting using chan.Attenuator (which automatically sets AttenuatorMode = naManual.
Return Type	NAModes
Default	0 - Auto

Examples	chan.AttenuatorMode(1) = naAuto 'Write attn = chan.AttenuatorMode(1) 'Read
C++ Syntax	HRESULT get_AttenuatorMode(long port, tagNAModes* pVal) HRESULT put_AttenuatorMode(long port, tagNAModes newVal)
Interface	IChannel

Write/Read **About Attenuation**

Attenuator Property

Description	Sets or returns the value of the attenuator control for the specified port number. Sending this command automatically sets AttenuatorMode to Manual.
VB Syntax	*chan*.**Attenuator(***portNum***)** = *value*
Vaariable	**(Type) - Description**
chan	A Channel **(object)**
portNum	**(long integer)** - Port number (**1** or **2**) of attenuator control to be changed.
value	**(double)** - Attenuator value in dB in 10dB steps. Choose any Long Integer between **0** and **70** If an invalid value is entered, the analyzer will select the next lower valid value. For example, if 19.9 is entered the analyzer will select 10 dB attenuation.
Return Type	Double
Default	20 dB
Examples	chan.Attenuator(1) = 20 'Write attn = chan.Attenuator(cnum) 'Read
C++ Syntax	HRESULT get_Attenuator(long port, double *pVal) HRESULT put_Attenuator(long port, double newVal)
Interface	IChannel

Write/Read **About Averaging**

Averaging Property

Description	Turns trace averaging ON or OFF for all measurements on the channel. Averaging is only allowed on ratioed measurements; not on single input measurements.
VB Syntax	*chan*.**Averaging** = *state*
Variable	**(Type) - Description**
chan	A Channel **(object)**
state	**(boolean)** **0** - Turns averaging OFF

	1 - Turns averaging ON
Return Type	Boolean
Default	0

Examples	chan.Average = 1 'Write averg = chan.Averaging 'Read

C++ Syntax	HRESULT get_Averaging(BOOL *pVal) HRESULT put_Averaging(BOOL newVal)
Interface	IChannel

Read-only **About Averaging**

AveragingCount Property

Description	Returns the number of sweeps that have been acquired and averaged into the measurements on this channel. AveragingFactor specifies the number of sweeps to average. AveragingCount indicates the progress toward that goal.
VB Syntax	*value = chan.***AveragingCount**

Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Long Integer)** - Variable to store the returned count
Return Type	Long Integer
Default	Not Applicable

Example	avgcount = chan.AveragingCount

C++ Syntax	HRESULT get_AveragingCount(long* count)
Interface	IChannel

Write/Read **About Averaging**

AveragingFactor Property

Description	Specifies the number of measurement sweeps to combine for an average. Must also turn averaging ON by setting *chan.*Averaging = 1. Averaging is only allowed on ratioed measurements; not on single input measurements.
VB Syntax	*chan.***AveragingFactor** = *value*

Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Long Integer)** - Number of measurement sweeps to average. Choose any number between **1** and **1024.**
Return Type	Long Integer
Default	1

Examples	chan.AveragingFactor = 5 'Write

avgfact = chan.AveragingFactor ' doesn't work -Read

C++ Syntax HRESULT get_AveragingFactor(long *pVal)
HRESULT put_AveragingFactor(long newVal)

Interface IChannel

Write/Read **About Frequency**

CenterFrequency Property

Description Sets or returns the center frequency of the channel
or
Sets or returns the center frequency of the segment.

see also Measurement2 interface

VB Syntax *object*.**centerFrequency** = *value*

Variable **(Type) - Description**

object A Channel **(object)**
or
A Segment **(object)**

value **(double)** - Center frequency in Hertz. Choose any number between the **minimum** and **maximum** frequencies of the analyzer.

Return Type Double

Default Center of the frequency range

Examples chan.centerFrequency = 4.5e9 'sets the center frequency of a linear sweep for the channel object -Write
centfreq = chan.centerFrequency 'Read

C++ Syntax HRESULT get_CenterFrequency(double *pVal)
HRESULT put_CenterFrequency(double newVal)

Interface IChannel
ISegment

Read-only **About Channels**

ChannelNumber Property

Description Returns the Channel number of the Channel or Measurement object.

VB Syntax *object*.**ChannelNumber**

Variable **(Type) - Description**

object A Channel **(object)**
or
A Measurement **(object)**

Return Type	Long Integer
Default	Not applicable
Examples	chanNum = chan.ChannelNumber 'returns the channel number chanNum = meas.ChannelNumber 'returns the channel number of the measurement
C++ Syntax	HRESULT get_ChannelNumber(long *pVal)
Interface	IChannel IMeasurement

Write/Read **About Power Coupling**

CouplePorts Property

Description	Turns ON and OFF port power coupling. ON means the power level is the same for both ports. OFF means the power level may be set independently for each port.
VB Syntax	*chan*.**CouplePorts** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(enum NAStates)** Choose from: **0 - NaOff** - Turns coupling OFF **1 - NaOn** - Turns coupling ON
Retaurn Type	Long Integer **1** - ON **0** - OFF
Default	NaON (1)
Examples	chan.CouplePorts = NaOff 'Write couplport = chan.CouplePorts 'Read
C++ Syntax	HRESULT get_CouplePorts(tagNAStates *pState) HRESULT put_CouplePorts(tagNAStates newState)
Interface	IChannel

Write/Read **About CW Frequency**

CW Frequency Property

Description	Set the Continuous Wave (CW) frequency. Must first send chan.SweepType = naCWTimeSweep
VB Syntax	*chan*.**CWFrequency** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(double)** CW frequency. Choose any number between: the **minimum** and **maximum** frequency limits of the analyzer Units are Hz

Return Type Double
Default 1e9

Examples

chan.CWFrequency = 5e9 'Write
cwfreq = chan.CWFrequency 'Read

C++ Syntax

HRESULT put_CWFrequency(double newVal)
HRESULT get_CWFrequency(double *pVal)

Interface IChannel

Write/Read **About Dwell Time**

DwellTime Property

Description

Sets or returns the dwell time at the start of each sweep point for all measurements in a channel. Dwell time is only available with Chan.SweepGenerationMode **= naSteppedSweep** (not naAnalogSweep).

Sets or returns the dwell time of a specified sweep segment.

VB Syntax *object*.**DwellTime** = *value*

Variable	**(Type) - Description**
object	A Channel **(object) or** A Segment **(object)**
value	**(double)** - Dwell Time in seconds. Choose any number between**:** **0** and **100e-3**

Return Type Double
Default 0

Examples

```
chan.DwellTime = 3e-3 'sets the dwell time for the channel -Write
```

segs(3).CenterFrequency = 1e9 'sets the dwell time of segment 3 -Write
dwell = chan.DwellTime 'Read

C++ Syntax

HRESULT get_DwellTime(double *pVal)
HRESULT put_DwellTime(double newVal)

Interface IChannel
ISegment

Write/Read **About Frequency Range**

FrequencySpan Property

Description

Sets or returns the frequency span of the channel.

Sets or returns the frequency span of the segment.

VB Syntax *object*.**FrequencySpan** = *value*

Variable **(Type) - Description**

object	A Channel **(object)** **or** A Segment **(object)**
value	**(double)** - Frequency span in Hertz. Choose any number between the **minimum** and **maximum** frequencies of the analyzer.
Return Type	Double
Default	Full frequency span of the analyzer
Examples	chan.FrequencySpan = 4.5e9 'sets the frequency span of a linear sweep for the channel object -Write freqspan = chan.FrequencySpan 'Read
C++ Syntax	HRESULT get_FrequencySpan(double *pVal) HRESULT put_FrequencySpan(double newVal)
Interface	IChannel ISegment

Write/Read **About Frequency Offset**

FrequencyOffsetCWOverride Property

Description	Establishes a fixed (CW) stimulus frequency while measuring the Response over a swept frequency range. For example, a fixed-frequency PNA stimulus may be applied to the RF input of a mixer whose local oscillator (LO) is being swept. Because the IF output of the mixer will be swept, the PNA receivers must also be swept. See other Frequency Offset properties.
VB Syntax	*chan*.**FrequencyOffsetCWOverride** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Enum as NaStates)** - Choose from: naOFF (0) - Turns CW override OFF naON (1) - Turns CW override ON
Return Type	Enum
Default	0 Hz
Examples	chan.FrequencyOffsetCWOverride = 1 'Write fOffsetOV = chan.FrequencyOffsetCWOverride 'Read
C++ Syntax	HRESULT get_FrequencyOffsetCWOverride (tagNAStates *pstate) HRESULT put_FrequencyOffsetCWOverride (tag NAStates newState)
Interface	IChannel2

Write/Read **About Frequency Offset**

FrequencyOffsetDivisor Property

Description	Specifies (along with FrequencyOffsetMultiplier) the value to multiply by the stimulus. See other Frequency Offset properties
VB Syntax	*chan*.**FrequencyOffsetDivisor** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Double)** - Divisor value. Range is 1 to 1000
Return Type	Double
Default	1
Examples	chan.FrequencyOffsetDivisor = 2 'Write fOffsetDiv = chan.FrequencyOffsetDivisor 'Read
C++ Syntax	HRESULT get_FrequencyOffsetDivisor(double*pval) HRESULT put_FrequencyOffsetDivisor(double newVal)
Interface	IChannel2

Write/Read **About Frequency Offset**

FrequencyOffsetFrequency Property

Description	Specifies an absolute offset frequency in Hz. For mixer measurements, this would be the LO frequency. See other Frequency Offset properties.
VB Syntax	*chan*.**FrequencyOffsetFrequency** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Double)** - Offset value. Range is +/- 1000 GHz. (Offsets can be positive or negative.)
Return Type	Double
Default	0 Hz
Examples	chan.FrequencyOffsetFrequency = 2 'Write fOffsetFreq = chan.FrequencyOffsetFrequency 'Read
C++ Syntax	HRESULT get_FrequencyOffsetFrequency(double*pval) HRESULT put_FrequencyOffsetFrequency(double newVal)
Interface	IChannel2

Write/Read **About Frequency Offset**

FrequencyOffsetMultiplier Property

Description	Specifies (along with FrequencyOffsetDivisor) the value to multiply by the stimulus. See other Frequency Offset properties.
VB Syntax	*chan*.**FrequencyOffsetMultiplier** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**

value	**(Double)** - Multiplier value. Range is 1 to 1000
Return Type	Double
Default	1

Examples	chan.FrequencyOffsetMultiplier = 2 'Write fOffsetMult = chan.FrequencyOffsetMultiplier 'Read

C++ Syntax	HRESULT get_FrequencyOffsetMultiplier (double*pval); HRESULT put_FrequencyOffsetMultiplier (double newVal);
Interface	IChannel2

Write/Read **About Frequency Offset**

FrequencyOffsetState Property

Description	Enables Frequency Offset on ALL measurements that are present on the active channel. This immediately causes the source and receiver to tune to separate frequencies. The receiver frequencies are specified with other channel and offset settings. To make the stimulus settings, use Channel Start, Stop Frequency properties. See other Frequency Offset properties. Tip: To avoid unnecessary errors, first make other frequency offset settings. Then turn Frequency Offset ON.
VB Syntax	*chan*.**FrequencyOffsetState** = *value*

Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Enum as NaStates)** - Choose from: **naOFF** (0) - Turns Frequency Offset OFF **naON** (1) - Turns Frequency Offset ON
Return Type	Enum
Default	naOFF (0)

Examples	chan.FrequencyOffsetState = True 'Write Foffset = chan.FrequencyOffsetState 'Read

C++ Syntax	HRESULT FrequencyOffsetState (tag NAStates *pState); HRESULT FrequencyOffsetState (tag NAStates newState)
Interface	IChannel2

Write/Read **About IF Bandwidth**

IFBandwidth Property

Description	Sets or returns the IF Bandwidth of the channel. Sets or returns the IF Bandwidth of the segment.
VB Syntax	*object*.**IFBandwidth** = *value*

Variable	**(Type) - Description**
object	A Channel **(object) or** A Segment **(object)**
value	**(double)** - IF Bandwidth in Hz. Choose from:

1 | 2 | 3 | 5 | 7 | 10 | 15 | 20 | 30 | 50 | 70 | 100 | 150 | 200 | 300 | 500 | 700 | 1000 | 1500 | 2000 | 3000 | 5000 | 7000 | 10000 | 15000 | 20000 | 30000 | 35000 | 40000 |

If a number other than these is entered, the analyzer will round up to the closest valid number (unless a number higher than the maximum in entered.)

Return Type Double

Default 3500

Examples

chan.IFBandwidth = 3e3 'sets the IF Bandwidth of for the channel object to 3 kHz. -Write
seg.IFBandwidth = 5 'sets the IF Bandwidth of the segment to 5 Hz. -Write
ifbw = chan.IFBandwidth -Read

C++ Syntax

HRESULT get_IFBandwidth(double *pVal);
HRESULT put_IFBandwidth(double newVal);

Interface

IChannel
ISegment

Write/Read **About Number of Points**

NumberOfPoints Property

Description

Sets or returns the Number of Points of the channel.
Sets or returns the Number of Points of the segment.

see also Measurement2 interface

VB Syntax *object*.**NumberOfPoints** = *value*

Variable	**(Type) - Description**
object	A Channel **(object) or** A Segment **(object)**
value	**(long)** - Number of Points. For channel, choose any number from **1** to **16001.** For segment, the total number of points in all segments cannot exceed **16001.** A segment can have as few as 1 point.

Return Type Long Integer

Default

201 for channel
21 for segment

Examples

chan.NumberOfPoints = 201 'sets the number of points for all measurements in the channel. -Write
numofpts = chan.NumberOfPoints 'Read

C++ Syntax

HRESULT get_NumberOfPoints(long *pVal)
HRESULT put_NumberOfPoints(long newVal)

Interface

IChannel
ISegment

Write/Read **About Power Slope**

PowerSlope Property

Description	Sets or returns the Power Slope value. Power Slope function increases or decreases the output power over frequency. Units are db/GHz. For example: PowerSlope = 2 will increase the power 2db/1GHZ.
VB Syntax	*app*.**PowerSlope** = *value*
Variable	**(Type) - Description**
app	An Application **(object)**
value	**(double)** - Power Slope. Choose any number between -2 and 2. **No slope = 0**
Return Type	Double
Default	0
Examples	app.PowerSlope = 2 'Write pwrslp = app.PowerSlope 'Read
C++ Syntax	HRESULT get_PowerSlope(double *pVal) HRESULT put_PowerSlope(double newVal)
Interface	IChannel

Write/Read **About Receiver Attenuation**

ReceiverAttenuator Property

Description	Sets or returns the value of the specified receiver attenuator control.
VB Syntax	*chan*.**ReceiverAttenuator(***rec***)** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
rec	**(long integer)** - Receiver with attenuator control to be changed. Choose from: **1** - Receiver A **2** - Receiver B
value	**(double)** - Attenuator value in dB. Choose any Long Integer between 0 and 35 in 5dB steps: If an invalid value is entered, the analyzer will select the next lower valid value. For example, if 19.9 is entered the analyzer will select 15 dB attenuation.
Return Type	Double
Default	0 db
Examples	chan.ReceiverAttenuator(1) = 5 'Write attn = chan.ReceiverAttenuator(2) 'Read
C++ Syntax	HRESULT get_ReceiverAttenuator(long lport, double *pVal) HRESULT put_ReceiverAttenuator(long lport, double newVal)
Interface	IChannel

Write/Read **About Frequency Offset**

R1inputPath Property

Description	PNA models with option 081 have a switch in the test set that allows access to the port 1 reference receiver through the front panel Reference 1 connectors. This command throws that switch between the internal path to the receiver, or through the external connectors. You could use this feature to make converter measurements relative to a reference ("golden") mixer. See other Frequency Offset properties.
VB Syntax	chan.**R1InputPath** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Enum as naInputPath)** - Choose from:naPathInternal - **(0)** - internal path to the reference receiver naPathExternal **(1)** - path through external connectors
Return Type	Enum
Default	naPathInternal - (0)
Examples	`chan.R1InputPath = naPathInternal 'Write` Inpath = chan.R1InputPath 'Read
C++ Syntax	HRESULT get_R1InputPath (tag NAInputPath *pPath); HRESULT put_R1InputPath (tag NAInputPath newPath);
Interface	IChannel2

Write / Read **About Source Power Cal**

SourcePowerCorrection Property

Description	Sets source power correction ON or OFF for a specific source port on this channel, or returns the current ON or OFF state of correction for that source port.
VB Syntax	*chan*.**SourcePowerCorrection**(sourcePort) = *value*
Variable	**(Type) - Description**
chan	**(object)** – A Channel object
sourcePort	**(long integer)** – Source port for which to set or return the ON or OFF state of source power correction.
value	**(boolean)** False (0) – Turns source power correction OFF for the source port. True (1) – Turns source power correction ON for the source port.
Return Type	Boolean
Default	False (0) - Source power correction will turn correction ON

Examples chan.SourcePowerCorrection(1) = 1 'Write
calOnPort2 = chan.SourcePowerCorrection(2) 'Read

C++ Syntax HRESULT put_SourcePowerCorrection(VARIANT_BOOL bState);
HRESULT get_SourcePowerCorrection(VARIANT_BOOL *bState);

Interface IChannel

Write/Read **About Linear Frequency Sweep**

StartFrequency Property

Description Sets or returns the start frequency of the channel
or
Sets or returns the start frequency of the segment.
see also Measurement2 interface

VB Syntax *object*.**StartFrequency** = *value*

Variable **(Type) - Description**

object A Channel **(object)**
or
A Segment **(object)**

value **(double)** - Start frequency in Hertz. Choose any number between the **minimum** and **maximum** frequencies of the analyzer.

Return Type Double

Default Channel - Minimum frequency of the analyzer
Segment - 0

Examples chan.StartFrequency = 4.5e9 'sets the start frequency of a linear sweep for the channel object -Write
startfreq = Chan.StartFrequency 'Read

C++ Syntax HRESULT get_StartFrequency(double *pVal)
HRESULT put_StartFrequency(double newVal)

Interface IChannel
ISegment

Write/Read **About Power Sweep**

StartPower Property

Description Sets the start power of the analyzer when sweep type is set to Power Sweep. Frequency of the measurement is set with chan.CWFrequency.

VB Syntax *chan*.**StartPower** = *value*

Variable **(Type) - Description**

chan A Channel **(object)**

value **(double)** - Start Power in dBm. There is 40 dB of range in power sweep. The values of start and stop depend on the amount of attenuation that you specify. With 0 dB of attenuation, the range is -20 dBm to +20 dBm.

With 10 dB of attenuation, the range is -30 dBm to +10 dBm, and so forth. Auto attenuation is not allowed in Power Sweep.

Return Type Double

Default 0

Examples

Chan.StartPower = -10 'Write
strtpwr = Chan.StartPower 'Read

C++ Syntax

HRESULT get_StartPower(double *pVal)
HRESULT put_StartPower(double newVal)

Interface IChannel

Write/Read **About Linear Frequency Sweep**

StopFrequency Property

Description

Sets or returns the stop frequency of the channel
or
Sets or returns the stop frequency of the segment.

see also Measurement2 interface

VB Syntax *object*.**StopFrequency** = *value*

Variable	**(Type) - Description**
object	A Channel **(object)** **or** A Segment **(object)**
value	**(double)** - Stop frequency in Hertz. Choose any number between the **minimum** and **maximum** frequencies of the analyzer.

Return Type Double

Default Channel - Maximum frequency of the analyzer
Segment - 0

Examples

chan.StopFrequency = 4.5e9 'sets the stop frequency of a linear sweep for the channel object -Write
stopfreq = Chan.StopFrequency 'Read

C++ Syntax

HRESULT get_StopFrequency(double *pVal)
HRESULT put_StopFrequency(double newVal)

Interface IChannel
ISegment

Write/Read **About Power Sweep**

StopPower Property

Description Sets the Stop Power of the analyzer when sweep type is set to Power Sweep. Frequency of the measurement is set with chan.CWFrequency

VB Syntax *chan*.**StopPower** = *value*

Variable	**(Type) - Description**

chan	A Channel **(object)**
value	**(double)** - Stop Power in dB. Start Power in dB. There is 40 dB of range in power sweep. The acceptable values of start and stop depend on the amount of attenuation that you specify. With 0 dB of attenuation, the range is -20 dBm to +20 dBm. With 10 of attenuation, the range is -30 dBm to +10 dBm, and so forth. Auto attenuation is not allowed in Power Sweep.
Return Type	Double
Default	0
Examples	Chan.StopPower = -10 'Write stppwr = Chan.StopPower 'Read
C++ Syntax	HRESULT get_StopPower(double *pVal) HRESULT put_StopPower(double newVal)
Interface	IChannel

Write/Read **About Stepped Sweep**

SweepGenerationMode Property

Description	Sets the method used to generate a sweep: continuous ramp (analog) or discrete steps (stepped).
VB Syntax	*chan*.**SweepGenerationMode** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(enum NASweepGenerationModes)** - **Choose either:** **0 - naSteppedSweep** - source frequency is CONSTANT during measurement of eah displayed point. More accurate than Analog. Dwell time can be set in this mode. **1 - naAnalogSweep** - source frequency is continuously RAMPING during measurement of each displayed point. Faster than Stepped. Sweep time (not dwell time) can be set in this mode.
Return Type	Long Integer
Default	Analog
Examples	Chan.SweepGenerationMode = naAnalogSweep 'Write swpgen = Chan.SweepGenerationMode 'Read
C++ Syntax	HRESULT get_SweepGenerationMode(tagNASweepGenerationModes* pVal) HRESULT put_SweepGenerationMode(tagNASweepGenerationModes newVal)
Interface	IChannel

Write/Read **About Sweep Time**

SweepTime Property

Description	Sets the Sweep time of the analyzer. Sweep time is limited so that the analyzer only sweeps as fast as possible for the current frequency range, number of points, and IFbandwidth.
VB Syntax	*chan*.**SweepTime** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(double)** - Sweep time in seconds. Choose a number between: **0** and **100**
Return Type	Double
Default	0
Examples	chan.SweepTime = 3e-3 'Write swptme = chan.SweepTime 'Read
C++ Syntax	HRESULT get_SweepTime(double *pVal) HRESULT put_SweepTime(double newVal)
Interface	IChannel

Write/Read **About Sweep Types**

SweepType Property

Description	Sets the type of X-axis sweep that is performed on a channel.
VB Syntax	*chan*.**SweepType** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(enum NASweepTypes)** - Choose from: **0 - naLinearSweep** **1 - naLogSweep** **2 - naPowerSweep** **3 - naCWTimeSweep** **4 - naSegmentSweep** **Note**: Sweep type cannot be set to Segment sweep if there are no segments turned ON. A segment is automatically turned ON when a application is created.
Return Type	Long Integer
Default	naLinearSweep
Examples	chan.SweepType = naPowerSweep 'Write swptyp = chan.SweepType 'Read
C++ Syntax	HRESULT get_SweepType(tagNASweepTypes* pVal) HRESULT put_SweepType(tagNASweepTypes newVal)
Interface	IChannel

Write/Read **About Power Level**

TestPortPower Property

Description	Sets or returns the RF power level for the channel **or** Sets or returns the RF power level of the segment.
VB Syntax	*object*.**TestPortPower***(portNum)* = *value*
Variable	**(Type) - Description**
object	A Channel **(object) -** to set coupled power, use chan.CouplePorts. If CouplePorts = False, then each port power can be set independently. Otherwise, chanTestPortPower (1) = value sets power level at both ports. **or** A Segment **(object)**
portNum	**(long integer)** - Port number of the source power. Choose from **1** or **2**
value	**(double)** - RF Power in dBm. Choose any number between **-90 and 20.** Actual achievable leveled power depends on frequency.
Return Type	Double
Default	0
Examples	chan.TestPortPower(1) = 5 'sets the port 1 RF power level for the channel object -Write powerlev = Chan.TestPortPower(1) 'Read
C++ Syntax	HRESULT get_TestPortPower(long port, double *pVal) HRESULT put_TestPortPower(long port, double newVal)
Interface	IChannel ISegment

Write/Read **About Triggering**

TriggerMode Property

Description	Each trigger signal will cause either: all measurements in the channel to be made **or** only a single data point in the channel at a time.
VB Syntax	*chan.***TriggerMode** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(enum NATriggerMode)** - Choose from: **0 - naTriggerModePoint** - a single data point is measured with each trigger signal the channel receives. Subsequent trigger signals continue to go to the channel in Point mode until the channel measurements are complete. **1 - naTriggerModeMeasurement** - all measurements in the channel are made with each trigger signal the channel receives. **Note:** Point Mode is not compatible when TriggerType is set to naGlobalTrigger. If you change any channel to TriggerModePoint, TriggerType will be set to naChannelTrigger.
Return Type	Long Integer
Default	0 - naTriggerModeMeasurement

Examples chan.TriggerMode = naTriggerModePoint 'Write
trigtyp = chan.TriggerMode 'Read

C++ Syntax HRESULT get_TriggerMode (tagNATriggerMode *pMode)
HRESULT put_TriggerMode (tagNATriggerMode newMode)

Interface IChannel

Write/Read **About User Ranges**

UserRangeMax Property

Description Sets the stimulus stop value for the specified User Range.

This property uses different arguments for the channel and marker objects.

VB Syntax *chan*.**UserRangeMax(***domainType,Mnum***)** = *value*
or
mark.**UserRangeMax(***rnum***)** = *value*

Variable	(Type) - Description
chan	A Channel **(object)**
mark	A Marker **(object)** To assign a marker to a User Range, use the UserRange Property. **Note:** The Marker object does not require the "DomainType" argument.
domainType	**(enum NADomainType)** - Choose from: **0 - naDomainFrequency** **1 - naDomainTime** **2 - naDomainPower**
Mnum	**(long integer)** - User Range number. Choose any number between 1 and 9 (0=Full Span)
value	**(double)** - Stop value. Choose any number within the full span of the channel

Return Type Double

Default The current stimulus setting for the channel

Examples mark.UserRangeMax(1) = 3e9 'Write
chan.UserRangeMax(naDomainFrequency,1) = 3e9 'Write
UseRngeMax = mark.UserRangeMax 'Read
UseRngeMax = chan.UserRangeMax 'Read

C++ Syntax HRESULT put_UserRangeMax(tagNADomainType domain, long rangeNumber, double maxValue)
HRESULT get_UserRangeMax(tagNADomainType domain, long rangeNumber, double *maxValue)

Interface IChannel

Write/Read **About User Ranges**

UserRangeMin Property

Description	Sets the stimulus start value for the specified User Range. This property uses different arguments for the channel and marker objects.
VB Syntax	*chan*.**UserRangeMin(***domainType,range***)** = *value* **or** *mark*.**UserRangeMin(***range***)** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
mark	A Marker **(object)** To assign a marker to a User Range, use the UserRange Property. **Note:** The Marker object does not require the DomainType argument
domainType	**(enum NADomainType)** Type of sweep currently implemented on the channel - Choose from: **0 - naDomainFrequency** **1 - naDomainTime** **2 - naDomainPower**
range	**(long)** - User Range number. Choose any number between **1** and **9** (0=Full Span)
value	**(double)** - Start value. Choose any number within the full span of the analyzer
Return Type	Double
Default	The current stimulus setting for the channel
Examples	mark.UserRangeMin(1) = 3e9 'Write chan.UserRangeMin(naDomainFrequency,1) = 3e9 'Write UseRngeMin = mark.UserRangeMin 'Read UseRngeMin = chan.UserRangeMin 'Read
C++ Syntax	HRESULT put_UserRangeMin(tagNADomainType domain, long rangeNumber, double minValue) HRESULT get_UserRangeMin(tagNADomainType domain, long rangeNumber, double *minValue)
Interface	IChannel

Write/Read **About X-Axis Spacing**

XAxisPointSpacing Property

Description	Sets X-axis Point Spacing for the displaytraces measured with segment sweeps on the active channel.
VB Syntax	*chan*.**XAxisPointSpacing** = *value*
Variable	**(Type) - Description**
chan	A Channel **(object)**
value	**(Enum as naStates)** - Choose from: **0 - naOFF** - Turns X-axis Point Spacing OFF **1 - naON** - Turns X-axis Point Spacing ON
Return Type	Enum
Default	0 - naOFF

Examples	chan.XAxisPointSpacing = naOFF 'Write xspac = chan.XAxisPointSpacing 'Read
C++ Syntax	HRESULT get_XAxisPointSpacing (tagNAStates *pState); HRESULT put_XAxisPointSpacing (tagNAStates newState);
Interface	IChannel2

Channel Object custom ISourcePowerCalData Interface

Read-only **About Source Power Cal**

getSourcePowerCalDataScalar Method

Description	Retrieves (as scalar values) requested source power calibration data, if it exists, from this channel. **Note**: This method exists on a non-default interface. If you cannot access this method, use the getSourcePowerCalData Method on IChannel.
VB Syntax	*chandata*.**getSourcePowerCalDataScalar** *sourcePort, numValues, data*
Variable	**(Type) - Description**
chandata	**(interface)** – – An ISourcePowerCalData interface pointing to a Channel (object)
sourcePort	**(long integer)** – The source port for which calibration data is being requested.
numValues	**(long integer)** – Number of data values. [out] – specifies number of data values returned. [in] – specifies number of values being requested (this must not be larger than the capacity of the data array).
data	**(single)** – Array to store the data.
Return Type	Single
Default	Not Applicable

Examples

```
Dim numValues As Long
 Dim scalarCalValues() As Single
 Dim chanData As ISourcePowerCalData
 Const port1 As Long = 1
 numValues = app.ActiveChannel.NumberOfPoints
 ReDim scalarCalValues(numValues)
 Set chanData = app.ActiveChannel

 chanData.getSourcePowerCalDataScalar port1, numValues,
scalarCalValues(0)

 'Print the data
 For i = 0 to numValues - 1
 Print scalarCalValues(i)
 Next I
```

C++ Syntax HRESULT getSourcePowerCalDataScalar(long sourcePort, long

	*pNumValues, float *pVals);
Interface	ISourcePowerCalData

Write-only. **About Source Power Cal**

putSourcePowerCalDataScalar Method

Description	Inputs source power calibration data (as scalar values) to this channel for a specific source port.
VB Syntax	*chandata*.**putSourcePowerCalDataScalar** *sourcePort, numValues, data*
Variable	**(Type) - Description**
chandata	**(interface)** – – An ISourcePowerCalData interface pointing to a Channel (object)
sourcePort	**(long integer)** – The source port for which calibration data is being input.
numValues	**(long integer)** – Number of data values being input. **Note:** If this does not equal the current number of points on the channel, the calibration will not be valid.
data	**(single)** – Array of source power cal data being input.
Return Type	None
Default	Not Applicable
Examples	Dim chanData As ISourcePowerCalData Set chanData = app.ActiveChannel chanData.putSourcePowerCalDataScalar 1, 201, scalarCalValues(0)
C++ Syntax	HRESULT putSourcePowerCalDataScalar(long sourcePort, long numValues, float *pVals);
Interface	ISourcePowerCalData

Channels Collection

Channels Collection

Description

A collection object that provides a mechanism for iterating through the channels

Collections are, by definition, unordered lists of like objects. You cannot assume that Channels.Item(1) is always Channel 1. For more information, see Collections in the Analyzer.

Methods	**Description**
Add	An alternate way to create a measurement.
Item	Use to get a handle on a channel in the collection.
Properties	**Description**
Count	Returns the number of channels in the analyzer.
Parent	Returns a handle to the current Application.

Write-only **About Channels**

Add (channels) Method

Description	Creates a channel and returns a handle to it. If the channel already exists, it returns the handle to the existing channel.
VB Syntax	*chans.***Add** *(item)*
Variable	**(Type) - Description**
chans	A Channel collection **(object)**
item	**(variant)** - Channel number.
Return Type	Channel
Default	Not Applicable
Examples	chans.Add 3 'Creates channel 3
C++ Syntax	HRESULT Add(VARIANT numVal, IChannel** pChannel)
Interface	IChannels

Gating Object

Gating Object (default interface is IGating)

Description

Contains the methods and properties that control Time Domain Gating.

Methods

None

Property	**Description**
Center	Sets or returns the Center time. Shared with the Transform Object
Shape	Specifies the shape of the gate filter.
Span	Sets or returns the Span time. Shared with the Transform Object
Start	Sets or returns the Start time. Shared with the Transform Object
State	Turns an Object ON and OFF.
Stop	Sets or returns the Stop time. Shared with the Transform Object
Type	Specifies the type of gate filter used.

Write/Read **About Gating**

Center Property

Description	Sets or returns the Center time of either Gating or Time Domain transform windows
VB Syntax	*object*.**Center** = *value*
Variable	**(Type) - Description**
object	**(object)** As Gating **or** **(object)** As Transform
value	**(double)** - Center time in seconds. Choose any number between: **± (points-1) / frequency span**
Return Type	Double
Default	0
Examples	trans.Center = 4.5e-9 'sets the Center time of a transform window -Write gate.Center = 4.5e-9 'sets the Center time of a gating window -Write cnt = trans.Center 'Read
C++ Syntax	HRESULT get_Center(double *pVal) HRESULT put_Center(double newVal)
Interface	ITransform IGating

Write/Read **About Gate Filter**

Shape Property

Description	Specifies the shape of the gate filter.
VB Syntax	*gat*.**Shape** = *value*
Variable	**(Type) - Description**
gat	A Gating **(object)**
value	**(enum** NAGateShape**)** - Choose from: **0** - naGateShapeMaximum **1** - naGateShapeWide **2** - naGateShapeNormal **3** - naGateShapeMinimum
Return Type	NAGateShape
Default	2 - Normal
Examples	gat.Shape = naGateShapeMaximum 'Write filterShape = gat.Shape 'Read
C++ Syntax	HRESULT get_Shape(tagNAGateShape *pVal) HRESULT put_Shape(tagNAGateShape newVal)
Interface	IGating

Write/Read **About Time Domain**

Span Property

Description	Sets or returns the Span time of either Gating or Time Domain transform windows
VB Syntax	*object*.**Span** = *value*
Variable	**(Type) - Description**
object	**(object)** As Gating or **(object)** As Transform
value	**(double)** - Span time in seconds. Choose any number between: **2*[(number of points-1) / frequency span]** and **0**
Return Type	Double
Default	20ns
Examples	Trans.Span = 4.5e-9 'sets the time span of a transform window -Write Gate.Span = 4.5e-9 'sets the Span time of a gating window -Write span = Trans.Span 'Read
C++ Syntax	HRESULT get_Span(double *pVal) HRESULT put_Span(double newVal)
Interface	ITransform IGating

Write/Read **About Time Domain**

Start Property

Description	Sets or returns the start time of either Gating or Time Domain transform windows
VB Syntax	*object*.**Start** = *value*
Variable	**(Type) - Description**
object	**(object)** As Gating **or** **(object)** As Transform
value	**(double)** - Start time in seconds. Choose any number between: **± (number of points-1) / frequency span**
Return Type	Double
Default	-10ns
Examples	Trans.Start = 4.5e-9 'sets the start time of a transform window -Write Gate.Start = 4.5e-9 'sets the start time of a gating window -Write strt = Trans.Start 'Read
C++ Syntax	HRESULT get_Start(double *pVal) HRESULT put_Start(double newVal)
Interface	ITransform IGating

Write/Read | About Time Domain

Stop Property

Description	Sets or returns the Stop time of either Gating or Time Domain transform windows
VB Syntax	*object*.**Stop** = *value*
Variable	**(Type) - Description**
object	**(object)** As Gating **or** **(object)** As Transform
value	**(double)** - Start time in seconds. Choose any number between: **± (number of points-1) / frequency span**
Return Type	Double
Default	10 ns
Examples	Trans.Stop = 4.5e-9 'sets the stop time of a transform window -Write Gate.Stop = 4.5e-9 'sets the stop time of a gating window -Write stp = Trans.Stop 'Read
C++ Syntax	HRESULT get_Stop(double *pVal) HRESULT put_Stop(double newVal)
Interface	ITransform IGating

Write/Read | About Time Domain

Type Property

Description	Specifies the type of gate filter used.
VB Syntax	*gat*.**Type** = *value*
Variable	**(Type) - Description**
gat	A Gating **(object)**
value	**(enum NAGateType)** - Choose from: **0 - naGateTypeBandpass - I**ncludes (passes) the range between the start and stop times. **1 - naGateTypeNotch** - Excludes (attenuates) the range between the start and stop times.
Return Type	NAGateType
Default	Bandpass
Examples	gate.Type = naGateTypeNotch 'Write filterType = gate.Type 'Read
C++ Syntax	HRESULT get_Type(tagNAGateType *pVal) HRESULT put_Type(tagNAGateType newVal)
Interface	IGating

HWauxIO Object

HWAuxIO Object (default interface is IHWAuxIO3)

Description

Contains the methods and properties that control the rear panel Auxiliary Input / Output connector.

See a Pinout of the Aux IO Connector

This HWAuxIO2 interface extends the HWAuxIO interface. Use this interface to read and set the output voltage mode for the specified output.

This HWAuxIO3 interface extends the HWAuxIO interface. This interface enables the analyzer to accept and understand inputs from a footswitch connected to it.

Method	**Interface**	**Description**
get_InputVoltage	IHWAuxIO	Reads the ADC input voltage
get_OutputVoltage	IHWAuxIO	Reads voltages on the DAC/Analog Output 1 and Output 2
get_OutputVoltageMode	**IHWAuxIO2**	Reads mode setting for either DAC output.
get_PortCData	IHWAuxIO	Reads a 4-bit value from Port C
put_OutputVoltage	IHWAuxIO	Writes voltages to the DAC/Analog Output 1 and Output 2
put_OutputVoltageMode	**IHWAuxIO2**	Writes mode setting for either DAC output.
put_PortCData	IHWAuxIO	Writes a 4-bit value to Port C
Property		**Description**
FootSwitch	**IHWAuxIO3**	Reads the Footswitch Input
FootswitchMode	IHWAuxIO	Determines the action that occurs when the footswitch is pressed.
PassFailLogic	IHWAuxIO	Sets and reads the logic of the PassFail line Shared with the HWMaterialHandler Object
PassFailMode	IHWAuxIO	Sets and reads the mode of the PassFail line Shared with the HWMaterialHandler Object
PassFailScope	IHWAuxIO	Sets and reads the scope of the PassFail line Shared with the HWMaterialHandler Object
PortCLogic	HWAuxIO	Sets and reads the logic mode of Port C
PortCMode	HWAuxIO	Sets and reads the mode of Port C
SweepEndMode	HWAuxIO	Sets and reads the event that causes the Sweep End line to go to a false state. Shared with the HWMaterialHandler Object

Read-only **About the Aux I/O Connector**

get_InputVoltage Method

Description	Reads the ADC input voltage from Analog IN (pin 14) of the AUX IO connector
VB Syntax	*volts = AuxIO*.**get_InputVoltage**
Variable	**(Type) - Description**
volts	**(double) -** variable to store the return value
AuxIO	**(object) -** A Hardware Auxiliary Input / Output object
Return Type	Double
Default	0
Examples	**`Dim aux as HWAuxIO`** Set aux = PNA.getAuxIO volts = aux.get_InputVoltage 'read voltage on Analog In (pin 14)
C++ Syntax	HRESULT get_InputVoltage (double* Voltage);
Interface	HWAuxIO

Read-only **About the Aux I/O Connector**

get_OutputVoltage Method

Description	Reads voltages on the DAC/Analog Output 1 and Output 2 (pins 2 and 3 of the Aux I/O connector)
VB Syntax	*volts = AuxIO*.**get_OutputVoltage** *(output)*
Variable	**(Type) - Description**
volts	**(double) -** variable to store the return value
AuxIO	**(object) -** A Hardware Auxiliary Input / Output object
output	**(variant)** Number of the output DAC to read voltage from. Choose from: **1** - Output DAC 1 -(pin 3) **2 -** Output DAC 2 -(pin 2)
Return Type	Double
Default	
Examples	**`Dim aux as HWAuxIO`** **`Set aux = PNA.getAuxIO`** volts = aux.get_OutputVoltage(1) 'read voltage from Analog Out 1 (pin3)
C++ Syntax	HRESULT get_OutputVoltage(VARIANT Output, double* Voltage);
Interface	IHWAuxIO

Read-only

get OutputVoltageMode Method

Description	This command sets the mode of the selected "Analog Out" line on the Auxiliary IO. The modes give the user the option to have the requested voltage applied immediately or not until the sweep is done.Also see the description for "Analog Out 1, 2" in the Auxiliary IO connector documentation.

VB Syntax	vOutput2Mode = auxIo.get_OutputVoltageMode 2
Variable	**(Type) - Description** **(enum NAOutputVoltageMode)** **naWaitEOS** - While in this mode any voltage changes sent to the selected analog out will only get applied to the output between sweeps. **naNoWait** - While in this mode any voltage changes sent to the selected analog out will occur right away without waiting until the end of a sweep, the voltage gets applied immediately.
Return Type	**NAOutputVoltageMode**
Default	**naWaitEOS**
Examples	vOutput2Mode = auxIo.get_OutputVoltageMode 2 'Read
C++ Syntax	NAOutputVoltageMode *pVOutput1Mode; HRESULT hr = auxIo ->get_FootSwitchMode(1, pVOutput1Mode); // Read
Interface	IHWAuxIO

Read-only **About the Aux I/O Connector**

get_PortCData Method

Description	Reads a 4-bit value from Port C of the Aux I/O connector (pins 22-25) and the Material Handler IO (pins 21-24 Anritsu) - (pins 22-25 Avantest). **Note**: These lines are connected to both the Handler IO and Aux IO in the PNA.
VB Syntax	*value = AuxIO*.**get_PortCData**
Variable	**(Type) - Description**
value	**(variant)** - Variable to store the returned data
AuxIO	**(object) -** A Hardware Auxiliary Input / Output object
Return Type	Integer
Default	None
Examples	value = auxIo.get_PortCData 'Reading a value of 15 when in Positive Logic indicates Port C lines C0, C1, C2, C3 are High. If in Negative Logic they are Low.
C++ Syntax	HRESULT get_PortCData(VARIANT* Data);
Interface	IHWAuxIO

Write-only **About the Aux I/O Connector**

put_OutputVoltage Method

Description	Writes voltages on the DAC/Analog Output 1 and Output 2 (pins 2 and 3 of the Aux I/O connector)
VB Syntax	*AuxIO*.**put_OutputVoltage** *output, voltage*

Variable	(Type) - Description
AuxIO	**(object) -** A Hardware Auxiliary Input / Output object
output	**(variant)** Number of the output DAC to write voltage to. Choose from: **1** Output DAC 1 - (pin 2) **2** Output DAC 2 - (pin 3)
voltage	**(double)** Voltage to write to the output DAC. Choose a voltage from -10 to 10
Return Type	None
Default	None

Examples HWAuxIO.put_OutputVoltage 1,9 'set Analog Out1 to +9v

C++ Syntax HRESULT put_OutputVoltage (VARIANT Output, double Voltage);

Interface IHWAuxIO

Write-only

put_OutputVoltageMode Method

Description This command sets the mode of the selected "Analog Out" line on the Auxiliary IO. The modes give the user the option to have the requested voltage applied immediately or not until the sweep is done.Also see the description for "Analog Out 1, 2" in the Auxiliary IO connector documentation.

VB Syntax auxIo.put_OutputVoltageMode 1, naNoWait

Variable **(Type) - Description**

(enum NAOutputVoltageMode)

naWaitEOS - While in this mode any voltage changes sent to the selected analog out will only get applied to the output between sweeps.

naNoWait - While in this mode any voltage changes sent to the selected analog out will occur right away without waiting until the end of a sweep, the voltage gets applied immediately.

Return Type **NAOutputVoltageMode**

Default **naWaitEOS**

Examples

```
auxIo.put_OutputVoltageMode 1, naWaitEOS   'Write
```

auxIo.put_OutputVoltageMode 1, naNoWait 'Write

C++ Syntax NAOutputVoltageMode nuVOutput1Mode; nuVOutput1Mode = naWaitEOS; HRESULT hr = auxIo ->put_OutputVoltageMode (1, nuVOutput1Mode); // Write

Interface IHWAuxIO

put_PortCData Method

Description	Writes a 4-bit value to Port C on the Aux I/O connector (pins 22-25) and the Material Handler IO (pins 21-24 Anritsu) - (pins 22-25 Avantest). **Note**: These lines are connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.
VB Syntax	*AuxIO*.**put_PortCData** *num*
Variable	**(Type) - Description**
AuxIO	**(object) -** A Hardware Auxiliary Input / Output object
num	**(variant)** - 4 bit binary value. Choose from 0-15
Return Type	None
Default	None
Examples	HWAuxIO.put_PortCData 15 'If Positive Logic, Port C lines C0, C1, C2, C3 go High. If Negative Logic, they go Low.
C++ Syntax	HRESULT put_PortCData(VARIANT Data);
Interface	IHWAuxIO

Read-only

FootSwitch Property

Description	Reads the Footswitch Input (pin 20 of the AUX IO connector).
VB Syntax	*value = AuxIO*.**Footswitch**
Variable	**(Type) - Description**
value	**(boolean) -** Variable to store the returned value **False (0)** -foot switch is released **True (1)** - footswitch is depressed
AuxIO	**(object) -** A Hardware Aux I/O object
Return Type	Boolean
Default	True (1)
Examples	fs = aux.Footswitch
C++ Syntax	HRESULT get_FootSwitch (VARIANT_BOOL* State);
Interface	IHWAuxIO3

Write/Read **About the Aux I/O Connector**

FootswitchMode Property

Description	Determines what occurs when the footswitch is pressed. For more information see the FootSwitch In pin description in the Auxiliary IO connector.

VB Syntax	*AuxIo.***FootSwitchMode** = *value*

Variable	**(Type) - Description**
value	**(enum NAFootSwitchMode)** **0 - naIgnoreFootswitch** - Footswitch presses are ignored. **1 - naSweepTrigger** - Footswitch presses trigger a sweep. The PNA must be in Manual Trigger Mode. **2 - naRecallNextState** - Footswitch presses recall an instrument state. When more than one state is available, then each footswitch press recalls the next state, then starts over from the beginning. It is possible for a recalled state to override the current mode. If the recalled state is IGNore, then mode changes and additional footswitch presses are ignored. **3 - naRunMacro** - Footswitch presses load and run a macro. When more than one macro is available, then each footswitch press loads and runs the next macro, then starts over from the beginning. It is possible for a Macro to override the current mode. If the macro contains a Preset, then the mode changes to the default setting IGNore and additional footswitch presses are ignored.
AuxIO	**(object) -** A Hardware Aux I/O object
Return Type	**NAFootSwitchMode**
Default	**0 - naIgnoreFootswitch**

Examples	auxIo.FootSwitchMode = naIgnoreFootSwitch 'Write

C++ Syntax	HRESULT get_FootSwitchMode(NAFootSwitchMode *pFootSwitchMode) HRESULT put_FootSwitchMode(NAFootSwitchMode newFootSwitchMode)
Interface	IHWAuxIO3

Read/Write

PassFailLogic Property

Description	Sets and reads the logic of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12). **Note**: This line is connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.
VB Syntax	*object.***PassFailLogic** = *value*

Variable	**(Type) - Description**
object	**(object)** - An Aux I/O or Handler I/O object
value	**(enum as NARearPanelIOLogic)** Choose from: **0 - naPositiveLogic -** Causes the PassFail line to have positive logic (high = pass, low = fail). **1 - naNegativeLogic** - Causes the PassFail line to have negative logic (high = fail, low = pass).
Return Type	Long Integer

Default	naPositiveLogic
Examples	**`aux.PassFailLogic = naNegativeLogic 'Write`** Text1.Text = aux.PassFailLogic 'Read
C++ Syntax	HRESULT put_PassFailLogic (tagNARearPanelIOLogic Mode); HRESULT get_PassFailLogic (tagNARearPanelIOLogic* Mode);
Interface	IHWAuxIO IHWMaterialHandlerIO

Read/Write

PassFailMode Property

Description	Sets and reads the mode of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12). **Note**: This line is connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.
VB Syntax	*object*.**PassFailMode** = *value*
Variable	**(Type) - Description**
object	**(object)** - An Aux I/O or Handler I/O object
value	**(enum as NAPassFailMode)**.Choose from: **0 - naDefaultPassNoWaitMode**- the line stays in PASS state. When a device fails, then the line goes to fail IMMEDIATELY. **1 - naDefaultPassWaitMode** - the line stays in PASS state. When a device fails, then the line goes to fail after the Sweep End line is asserted. **2 - naDefaultFailWaitMode**- the line stays in FAIL state. When a device passes, then the line goes to PASS state after the Sweep End line is asserted.
Return Type	Long Integer
Default	0 - naDefaultPassNoWaitMode
Examples	**`HWAuxIO.PassFailMode = naDefaultPassNoWaitMode 'Write`** mode = HWAuxIO.PassFailMode 'Read
C++ Syntax	HRESULT put_PassFailMode (tagNAPassFailMode Mode); HRESULT get_PassFailMode (tagNAPassFailMode* Mode);
Interface	IHWAuxIO IHWMaterialHandlerIO

Read/Write

PassFailScope Property

Description

Sets and reads the Scope of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12).

Note: The PassFail line is connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.

VB Syntax

object.**PassFailScope** = *value*

Variable — **(Type) - Description**

object — **(object)** - An Aux I/O or Handler IO object

value — **(enum NAPassFailScope)** Choose from:

0 - naChannelScope - The PassFail line returns to its default state before sweeps on the next channel start. (A channel measurement may require several sweeps.)

1 - naGlobalScope - The PassFail line returns to its default state before the sweeps for the next **triggerable** channel start.

The default state of the PassFail line before a measurement occurs and after a failure occurs is set by the PassFailMode property.

Return Type

enum NAPassFailScope

Default

1 - naGlobalScope

Examples

```
HWAuxIO.PassFailScope = naGlobalScope 'Write
scope = HWAuxIO.PassFailScope 'Read
```

C++ Syntax

HRESULT put_PassFailScope (tagNAPassFailScope Scope);

HRESULT get_PassFailScope (tagNAPassFailScope* Scope);

Interface

IHWAuxIO

IHWMaterialHandlerIO

Read/Write **About the Aux I/O Connector**

PortCLogic Property

Description

Sets and reads the logic mode of Port C on the AUX IO connector and the Handler IO connector.

Note: Port C lines are connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.

VB Syntax

AuxIO.**PortCLogic** = *value*

Variable — **(Type) - Description**

AuxIO — **(object) -** A Hardware Aux I/O object

value — (**Enum as NaRearPanelIOLogic**) - Choose from:

0 - naPositiveLogic - The associated data line goes **HIGH** when writing a 1 to a PortC bit.

1 - naNegativeLogic - The associated data line goes **LOW** when writing a 1 to a PortC bit.

When Port C is in Output/Write mode, a change in logic causes the output lines to change state immediately. For example, Low levels change to High levels.

When Port C is in Input/Read mode, a change in logic will not cause the lines to change, but data read from Port C will reflect the change in logic.

Return Type Enum

Default 1 - naNegativeLogic

Examples

```
auxIO.PortCLogic = value 'Write
value = auxIo.PortCLogic 'Read
```

C++ Syntax

HRESULT put_PortCLogic (tagNARearPanelIOLogic Mode);

HRESULT get_PortCLogic (tagNARearPanelIOLogic* Mode);

Interface IHWAuxIO

Read/Write **About the Aux I/O Connector**

PortCMode Property

Description

Sets and reads whether Port C is setup for writing or reading data on the AUX IO connector and the Handler IO connector.

Note: Port C lines are connected to both the Handler IO and Aux IO in the PNA. Therefore, this command will affect both of these connectors in the same way.

VB Syntax *AuxIO*.**PortCMode** = *value*

Variable	**(Type) - Description**
AuxIO	**(object) -** A Hardware Aux I/O object
value	**(enum as NaPortMode)** - Choose from: **0 - naInput** - set the port for reading **1 - naOutput** - set the port for writing

Return Type Enum as NaPortMode

Default 1 - naInput

Examples

```
auxIo.get_PortCMode = naInput 'Write
value = auxIo.get_PortCMode 'Read
```

C++ Syntax

HRESULT get_PortCMode(tagNAPortMode* pMode);

HRESULT put_PortCMode(tagNAPortMode pMode);

Interface IHWAuxIO

Read/Write

SweepEndMode Property

Description

Sets and reads the event that will cause the Sweep End line to go to a low state. The line will return to a high state after the appropriate calculations are complete.

Note: This line is connected to the following pins on the HANDLER IO

	connector and AUX IO connector in the PNA. Therefore, this command will affect both of these connectors in the same way.
VB Syntax	*object*.**SweepEndMode** = *value*

Variable	**(Type) - Description**
object	**(object) -** A HandlerIO or AuxIO object
value	**(enum as NASweepEndMode) Choose from:** **0 - naSweep -** the line goes low when **each sweep** is complete **1 - naChannelSweep -** the line goes low when all the sweeps for **each channel** is complete. **2 - naGlobalSweep -** the line goes low when **all sweeps** for **all triggerable channels** are complete.
Return Type	Long Integer
Default	**0 - naSweep**

Examples	`HWAuxIO.PassFailMode = naSweep 'Write` value = HWAuxIO.PassFailMode 'Read

C++ Syntax	HRESULT put_SweepEndMode (tagNASweepEndMode Mode); HRESULT get_SweepEndMode (tagNASweepEndMode* Mode);
Interface	IHWAuxIO IHWMaterialHandlerIO

HWExternalTestSetIO Object

HWExternalTestSetIO Object (default interface is IHWExternalTestSetIO)

Description

Contains the methods and properties that control the rear panel External Test Set Input / Output connector

Pinout for the External Test Set Connector

Method	**Description**
ReadData	Reads data and generates the appropriate timing signals
ReadRaw	Reads data, but does NOT generate appropriate timing signals
WriteData	Writes data and generates the appropriate timing signals
WriteRaw	Writes data, but does NOT generate the appropriate timing signals
Property	**Description**
Interrupt	Returns the state of the Interrupt line
SweepHoldOff	Returns the state of the Sweep Holdoff line

Read-only **About the ExtTestSetIO connector**

ReadData Method

Description	Reads a 13-bit data word from the specified address. Data is read using the AD0 through AD12 lines of the external test set connector. The instrument generates the appropriate timing signals. It automatically controls timing signals LDS, LAS and RLW to strobe the address, and then read the data, from the external test set. See the timing diagram for Address and Data I/O read.
VB Syntax	*value = ExtIO*.**ReadData** *(address)*
Variable	**(Type) - Description**
value	**(variant)** - Variable to store the returned data
ExtIO	**(object) -** An ExternalTestSetIO object
address	**(variant)** - address to read data from.
Return Type	Variant
Default	Not Applicable
Examples	value = ExtIO.ReadData (15)
C++ Syntax	HRESULT ReadData (VARIANT Address, VARIANT* Data);
Interface	IHWExternaTestSetIO

Read-only **About the ExtTestSetIO connector**

ReadRaw Method

Description

Reads a 16-bit value from the external test set. The 16-bit value is comprised of lines AD0 - AD12, Sweep Holdoff In and Interrupt In (inverted).

When this command is used the analyzer does NOT generate the appropriate timing signals; it simply reads the lines. The user needs to first use the WriteRaw method to do the initial setup. The RLW line (pin25) must be set to the appropriate level in order to read the test set connected.

Below is the format of data that is read with ReadRaw:

Pin	**Bit**	**Signal name**
22	0	AD0*
23	1	AD1*
11	2	AD2*
10	3	AD3*
9	4	AD4*
21	5	AD5*
20	6	AD6*
19	7	AD7*
6	8	AD8*
5	9	AD9*
4	10	AD10*

17	11	AD11*
3	12	AD12*
2	13	Sweep Holdoff In
13	14	Interrupt In (inverted internally)
na	15	Always Zero, grounded internally

*These lines are dependent on the state of RLW (pin25).
Writing a 0(low) to RLW will set lines AD0-AD12 to write mode.
Writing a 1(high) to RLW will set lines AD0-AD12 to read mode.

VB Syntax *value = ExtIO.***ReadRaw** *(address)*

Variable	**(Type) - Description**
value	**(variant)** - Variable to store the returned data
ExtIO	**(object) -** An External IO object
address	**(variant)** - Address to read data from
Return Type	Real
Default	Not Applicable

Examples value = ExtIO.ReadRaw (address)

C++ Syntax HRESULT ReadRaw(VARIANT* Input);
Interface IHWExternalTestSetIO

Write-only **About the ExtTestSetIO connector**

WriteData Method

Description Writes a 13-bit value to the specified address using the AD0 through AD12 lines of the external test set connector. The instrument generates the appropriate timing signals. It automatically controls timing signals LDS, LAS and RLW to strobe the address, then the data, to the external test set. See the timing diagram for Address and Data I/O read.

VB Syntax *ExtIO.***ReadData** *(address) = value*

Variable	**(Type) - Description**
ExtIO	**(object) -** An External IO object
address	**(variant)** - address to be written to.
value	**(variant)** - 13-bit word to write
Return Type	Not Applicable
Default	Not Applicable

Examples ExtIO.WriteData (15) = 12

C++ Syntax HRESULT WriteData(VARIANT Address, VARIANT Data);
Interface IHWExternaTestSetIO

WriteRaw Method

Description Writes a 16-bit value to the external test set connector lines AD0 - AD12, RLW, LAS and LDS. The analyzer does NOT generate the appropriate timing signals. The user has control of all 16 lines using this write method.

Note: When RLW (pin25) is set to 1 (high) it causes lines AD0 - AD12 to float. It disables their output latches and sets the hardware for reading. LDS and LAS are not affected by this behavior.

Below is the format of data that is written with WriteRaw:

* This Output will float if RLW (bit-13) is set high

Pin	Bit	Signal name
22	0	AD0*
23	1	AD1*
11	2	AD2*
10	3	AD3*
9	4	AD4*
21	5	AD5*
20	6	AD6*
19	7	AD7*
6	8	AD8*
5	9	AD9*
4	10	AD10*
17	11	AD11*
3	12	AD12*
25	13	RLW
24	14	LDS
8	15	LAS

VB Syntax *ExtIO*.**WriteRaw** *value*

Variable	(Type) - Description
ExtIO	**(object) -** An External IO object
value	**(variant)** - Data to be written
Return Type	Not Applicable
Default	Not Applicable

Examples ExtIO.WriteRaw 12

C++ Syntax HRESULT WriteRaw(VARIANT Output);

Interface IHWExternalTestSetIO

Read-only **About the ExtTestSetIO connector**

Interrupt Property

Description	Reads the boolean that represents the state of the Interrupt In line (pin 13) on the external test set connector.
VB Syntax	*value = ExtIO*.**Interrupt**
Variable	**(Type) - Description**
value	**(boolean)** - Variable to store the returned data
ExtIO	**(object) -** An ExternalTestSetIO object
Return Type	Boolean **False (0)** - indicates the line is being held at a TTL High **True (1)** - indicates the line is being held at a TTL Low
Default	Not Applicable
Examples	value = ExtIO.Interrupt
C++ Syntax	HRESULT get_Interrupt(VARIANT_BOOL* bValue);
Interface	IHWExternalTestSetIO

Read-only **About the ExtTestSetIO connector**

SweepHoldOff Property

Description	Returns a boolean that represents the state of SweepHoldoff line (pin2) of the External Test Set connector.
VB Syntax	*value = ExtIO*.**SweepHoldOff**
Variable	**(Type) - Description**
value	**(boolean)** - Variable to store the returned data
ExtIO	**(object) -** An External IO object
Return Type	Boolean **False (0)** - indicates the line is being held at a TTL Low **True (1)** - indicates the line is being held at a TTL High
Default	Not Applicable
Examples	value = ExtIO.SweepHoldOff
C++ Syntax	HRESULT get_SweepHoldOff(VARIANT_BOOL* bValue);

Interface	IHWExternaTestSetIO

HWMaterialHandlerIO Object

HWMaterialHandlerIO Object (default interface is IHWMaterialHandlerIO)

Description

Contains the methods and properties that control the rear panel Material Handler Input / Output connector See the Pinout for the Material HandlerIO Connector

Method	**Description**
get_Input1	Reads a hardware latch that captures low to high transition on Input1
get_Output	Returns the last value written to the selected output pin.
get_Port	Returns the value from the specified "readable" port.
put_Output	Writes a TTL HI or TTL Low to output pins 3 or 4.
put_Port	Writes a value to the specified port.
Property	**Description**
PassFailLogic	Sets and reads the logic of the PassFail line Shared with the HWAuxIO Object
PassFailMode	Sets and reads the mode for the PassFail line Shared with the HWAuxIO Object
PassFailScope	Sets and reads the scope for the PassFail line Shared with the HWAuxIO Object
PortLogic	Sets and returns the logic mode of data ports A-H
PortMode	Sets and returns whether Port C or Port D is used for writing or reading data
SweepEndMode	Sets and reads the event that cause the Sweep End line to go to a low state. Shared with the HWAuxIO Object

Read-only **About the Handler IO Connector**

get_Input1 Method

Description

Reads a hardware latch that captures low to high transition on Input1 of the Material Handler IO. Reading the latch causes it to reset and is ready for the next transition. The hardware latch is only capable of capturing one transition per query. Additional transitions are ignored until after the next query.

Momentarily grounding or driving Input1 low then high causes a transition to be detected and latched.

VB Syntax	*inp1* = *handlerIo*.**get_Input1**

Variable	**(Type) - Description**
inp1	**(variant) -** A variable to store the return value
handlerIo	**(object)** - A HandlerIO object
Return Type	Variant - **1 -** a low to high transition occurred at Input1 since the last time it was queried. **0** - no low to high transition occurred.
Default	0

Examples	input1 = handlerIo.get_Input1 'Read

C++ Syntax	HRESULT get_Input1 (VARIANT* Data);
Interface	IHWMaterialHandlerIO

Read-only **About the Handler IO Connector**

get_Output Method

Description	Returns the last value written to the selected output pin. Data is written using put_Output Method
VB Syntax	*data* = *handlerIo*.**get_Output** *(pin)*

Variable	**(Type) - Description**
data	**(variant) -** A variable to store the return value. The returned value will be one of the following: **0** - TTL Low **1** - TTL High
handlerIo	**(object)** - A HandlerIO object
pin	**(enum as NAMatHandlerOutput)** - output pin to read value from. Choose from: **naOutput1 (0)** **naOutput1User (1)** **naOutput2 (2)** **naOutput2User (3)**
Return Type	Variant
Default	Not Applicable

Examples	data = handlerIo.get_Output(naOutput1)

C++ Syntax	HRESULT get_Output (tagNAMatHandlerOutput Output, VARIANT* Data);
Interface	IHWMaterialHandlerIO

Read-only **About the Handler IO Connector**

get_Port Method

Description	Returns the value from the specified "readable" port.
VB Syntax	*data* = *handlerIo*.**get_Port (port)**

Variable	(Type) - Description
data	**(variant) -** A variable to store the return value. The following table shows what the returned data represents:

Port	MSB..LSB 8..0
C	C3...C0
D	D3...D0
E	D3...D0 + C3...C0

handlerIo	**(object)** - A HandlerIO object
port	**(enum as NAMatHandlerPort)** - port to get data from. Choose from: **naPortC - (2)** **naPortD - (3)** **naPortE - (4)** **Note**: Reading data from the Write-only ports (A,B,F,G,H) will return an error. Ports C and D must be put in Read mode before reading from C, D, or E using PortMode Property.
Return Type	Variant
Default	0

Examples data = handlerIo.get_Port(naPortC)

C++ Syntax HRESULT get_Port (tagNAMatHandlerPort Port, VARIANT* Data);
Interface IHWMaterialHandlerIO

Write-only **About the Handler IO Connector**

put_Output Method

Description Writes a TTL HI or TTL Low to output pins 3 or 4 of the Material Handler IO connector.

Each pin also has a latched output which is written to with USER. With the latched (USER) outputs, the value is not applied to the associated pin until a positive edge is detected at INPUT1 (pin 2).

VB Syntax *handlerIo*.**put_Output** *(pin***) =** *value*

Variable	(Type) - Description
handlerIo	**(object)** - A HandlerIO object
pin	**(enum as NAMatHandlerOutput)** - pin to write data to. Choose from: **naOutput1 - (0) -** pin3 **naOutput1User (1) -** pin3 latched (applied to pin 3 on positive edge of Input1-pin2) **naOutput2 (2) -** pin4 **naOutput2User (3) -** pin4 latched (applied to pin 4 on positive edge of Input1-pin2)
value	**(Variant)** Value to write to the selected pin. Choose from

0 - TTL LOW

1 - TTL HIGH

Return Type Not Applicable

Default 0

Examples handlerIo.put Output(naOutput1)= 1

C++ Syntax HRESULT put_Output (tagNAMatHandlerOutput Output, VARIANT Data);

Interface IHWMaterialHandlerIO

Write-only **About the Handler IO Connector**

put Port Method

Description Writes a value to the specified port. Use the get_Port Method to read the settings from the "readable" ports (C, D, E).

VB Syntax *handlerIo*.**put_Port** *(port***) =** *value*

Variable **(Type) - Description**

handlerIo **(object)** - A HandlerIO object

port **(enum as NAMatHandlerPort)** - port to put data into. Choose from:

naPortA - (0)

naPortB - (1)

naPortC - (2)

naPortD - (3)

naPortE - (4)

naPortF - (5)

naPortG - (6)

naPortH - (7)

value The number of the data bits to set. The following table shows what the *value* represents:

Note: When writing to port G, port C must be set to output mode

When writing to port H, both port C and port D must be set to output mode. Use Port Mode Property

Port	**Max allowable <num>**	**MSB...LSB 23..0**	
A	255	A7...A0	Write-only
B	255	B7...B0	Write-only
C	15	C3...C0	Read-Write
D	15	D3...D0	Read-Write
E	255	D3...D0 + C3...C0	Read-Write
F	65535	B7...B0 + A7...A0	Write-only
G	1048575	C3...C0 + B7...B0 + A7...A0	Write-only
H	16777215	D3...D0 + C3...C0 + B7...B0 + A7...A0	Write-only

Return Type Not Applicable

Default Not Applicable

Examples handlerIo.put Port(naPortB)= 15

C++ Syntax HRESULT put_Port (tagNAMatHandlerPort Port, VARIANT Data);
Interface IHWMaterialHandlerIO

Read/Write **About the Handler I/O Connector**

PortLogic Property

Description Sets and returns the logic mode of data ports A-H on the HandlerIO connector. Port C of the Handler IO is connected internally to the Port C of the Aux IO connector. Therefore, it will have the same logic mode.

VB Syntax *handler*.**PortLogic** = *value*

Variable **(Type) - Description**

handler **(object) -** A HandlerI/O object

value (**enum as NaRearPanelIOLogic**) - Choose from:

0 - naPositiveLogic - When a value of one is written, the associated line goes High

1 - naNegativeLogic - When a value of one is written, the associated line goes Low

For ports that are in output (write) mode, a change in logic causes the output lines to change state immediately. For example, Low levels change immediately to High levels.

For ports that are in input (read) mode (C,D,E only), a change in logic will be reflected when data is read from that port. For example, if a line read 0, the next read after a logic change will read 1.

Return Type Long Integer

Default 1 - naNegativeLogic

Examples

```
handler.PortLogic = value 'Write
value = handler.PortLogic 'Read
```

C++ Syntax

HRESULT put_PortLogic(tagNARearPanelIOLogic Mode);

HRESULT get_PortLogic(tagNARearPanelIOLogic* Mode);

Interface IHWMaterialHandlerIO

Read/Write **About the Handler I/O Connector**

PortMode Property

Description Sets and returns whether Port C or Port D is used for writing or reading data on the Handler IO connector. The Handler IO Port C is connected internally to the Port C of the Aux IO connector. Therefore, the Aux IO connector will have the same input/output mode.

VB Syntax *handler*.**PortMode** *(port)* = *value*

Variable **(Type) - Description**

handler	**(object) -** A Handler I/O object
port	**(enum as NAMatHandlerPort)** Port to be changed. Choose from: **2 -naPortC** **3- naPortD**
value	**(enum as NaPortMode)** - Choose from: **0 - naInput** - set the port for reading **1 - naOutput** - set the port for writing
Return Type	Long Integer
Default	1 - naInput
Examples	**`handler.PortMode(naPortC) = naInput 'Write`** value = handler.PortMode(naPortD) 'Read
C++ Syntax	HRESULT put_PortMode (tagNAMatHandlerPort Port, tagNAPortMode Mode); HRESULT get_PortMode (tagNAMatHandlerPort Port, tagNAPortMode* Mode);
Interface	IHWMaterialHandlerIO

Limit Test Collection

Limit Test Collection

Description

Child of the **Measurement** Object. A collection that provides a mechanism for iterating through the Measurement's LimitSegment objects (Limit Lines). The collection has 100 limit lines by default.

The only way to get a handle to an individual limit line is by using the LimitTest collection. You can either **1)** set the property directly, or 2) set a variable a limit line in the LimitTest collection.

Examples

```
1)LimitTest(4).BeginResponse=.5
 2)Set lim2 = Application.Measurement.LimitTest(4)
```

Methods	**Description**
GetTestResult	Retrieves the Pass/Fail results of the Limit Test (State).
Item	Use to get a handle on a limit line in the collection.
Properties	**Description**
Count	Returns the number of limit lines used in the measurement.
LineDisplay	Displays the limit lines on the screen.
SoundOnFail	Enables a beep on Limit Test fails.
State	Turns ON and OFF limit testing.

Read-only **About Limit Testing**

GetTestResult Method

Description

Returns the result of limit line testing. There are three ways to use this command:

- If neither optional parameter is specified, limit results for ALL data is returned.
- If one parameter is specified (*start*), the limit result for that data point is returned.

If both parameters are specified, limit results are returned beginning with *start*, and ending with (*start+size*)-1

VB Syntax *testRes* = *limts.***GetTestResult [***start,size***]**

Variable	**(Type) - Description**
testRes	**(enum NALimitTestResult)** - A dimensioned variable to store test results. If a limit line is not tested, a PASS is returned. 1 - naLimitTestResult_Fail 2 - naLimitTestResult_Pass
limts	A LimitTest **(object)**
start	**(long)** - Optional argument. A start data point number to return limit test results.
size	**(long)** - Optional argument. Number of data points from *start* to return limit test results.
Return Type	Long Integer
Default	Not Applicable

Examples

```
Dim testRes As NALimitTestResult
 testRes = limts.GetTestResult
 Select Case testRes

 Case 1
 Print "Fails"

 Case 2
 Print "Pass"

 End Select
```

C++ Syntax HRESULT GetTestResult(long lStart, long lSize, tagNALimitTestResult *pVal)

Interface ILimitTest

Write/Read **About Limits**

LineDisplay Property

Description

Turns the display of limit lines ON or OFF. To turn limit TESTING On and OFF, use State Property.

Note: Trace data must be ON to view limit lines

VB Syntax *limitst.***LineDisplay** = *state*

Variable	**(Type) - Description**
limitst	A LimitTest **(object)**
state	**(boolean)** **0** - Turns the display of limit lines OFF **1** - Turns the display of limit lines ON
Return Type	Long Integer
Default	**1** - ON
Examples	Limttest.LineDisplay = 1 'Write lineDsp = Limttest.LineDisplay 'Read
C++ Syntax	HRESULT get_LineDisplay(VARIANT_BOOL *pVal) HRESULT put_LineDisplay(VARIANT_BOOL newVal)
Interface	ILimitTest

Write/Read **About Limits**

SoundOnFail Property

Description	Turns ON or OFF the audio indicator for limit failures.
VB Syntax	*limitst.***SoundOnFail** = *state*
Variable	**(Type) - Description**
limitst	A LimitTest **(object)**
state	**(boolean)** **0** - Turns the sound OFF **1** - Turns the sound ON
Return Type	Long Integer
Default	**1** - ON
Examples	Limttest.SoundOnFail = 1 'Write sound = Limttest.SoundOnFail 'Read
C++ Syntax	HRESULT get_SoundOnFail(VARIANT_BOOL *pVal) HRESULT put_SoundOnFail(VARIANT_BOOL newVal)
Interface	ILimitTest

LimitSegment Object

LimitSegment Object (default interface is ILimitSegment)

Description

The LimitSegment object is an individual limit line. The only way to get a handle to an individual limit line is by using the LimitTest collection. You can either **1)** set the property directly, or **2)** set a variable a limit line in the LimitTest collection.

Examples

```
1)LimitTest(4).BeginResponse=.5
```

2)Set lim2=Application.Measurement.LimitTest(4)

Methods	**Description**
None	
Properties	**Description**
BeginResponse	Specifies the Y-axis value that corresponds with Begin Stimulus (X-axis) value.
BeginStimulus	Specifies the beginning X-axis value of the Limit Line.
EndResponse	Specifies the Y-axis value that corresponds with End Stimulus (X-axis) value.
EndStimulus	Specifies the End X-axis value of the Limit Line.
Type	Specifies the Limit Line type.

Write/Read **About Limits**

BeginResponse Property

Description	When constructing a limit line, specifies the amplitude value of the start of a limit segment.
VB Syntax	*limtseg*.**BeginResponse** = *value*
Variable	**(Type) - Description**
limtseg	A LimitSegment **(object)**
value	**(double)** - Amplitude value. No units
Return Type	Double
Default	0
Examples	Set limtseg = meas.LimitTest(1) limtseg.BeginResponse = 10 'Write BegResp = limtseg.BeginResponse 'Read
C++ Syntax	HRESULT get_BeginResponse(double *pVal) HRESULT put_BeginResponse(double newVal)
Interface	ILimitSegment

Write/Read **About Limits**

BeginStimulus Property

Description	When constructing a limit line, specifies the beginning X-axis value.
VB Syntax	*limtseg*.**BeginStimulus** = *value*
Variable	**(Type) - Description**
limtseg	A LimitSegment **(object)**
value	**(double)** - Stimulus value. No units
Return Type	Double
Default	0

Examples Set limtseg = meas.LimitTest(1)

```
Set limtseg = meas.LimitTest(1)
  limtseg.Type = naLimitSegmentType_Maximum
  limtseg.BeginStimulus = 3e9
  limtseg.EndStimulus = 4e9
  limtseg.BeginResponse = 10
  limtseg.EndResponse = 10
BegStim = limtseg.BeginStimulus 'Read
```

C++ Syntax HRESULT get_BeginStimulus(double *pVal)
HRESULT put_BeginStimulus(double newVal)

Interface ILimitSegment

Write/Read **About Limits**

EndResponse Property

Description When constructing a limit line, specifies the amplitude value at the end of the limit segment.

VB Syntax *limtseg.***EndResponse** = *value*

Variable	**(Type) - Description**
limts	A LimitSegment **(object)**
value	**(double)** - Y-axis value of the End Response limit. No units
Return Type	Double
Default	0

Examples

```
Set limtseg = meas.LimitTest(1)
 limtseg.EndResponse = 10 'Write
EndResp = limtseg.EndResponse 'Read
```

C++ Syntax HRESULT get_EndResponse(double *pVal)
HRESULT put_EndResponse(double newVal)

Interface ILimitSegment

Write/Read **About Limits**

EndStimulus Property

Description When constructing a limit line, specifies the stimulus value for the end of the segment.

VB Syntax *limtseg.***EndStimulus** = *value*

Variable	**(Type) - Description**
limtseg	A LimitSegment **(object)**
value	**(double)** - End Stimulus X-axis value. No units
Return Type	Double
Default	0

Examples	Set limtseg = meas.LimitTest(1) limtseg.EndStimulus = 8e9 'Write EndStim = limtseg.EndStimulus 'Read
C++ Syntax	HRESULT get_EndStimulus(double *pVal) HRESULT put_EndStimulus(double newVal)
Interface	ILimitSegment

Write/Read **About Limits**

Type (limit) Property

Description	Specifies the Limit Line type.
VB Syntax	*limt(index)*.**Type** = value
Variable	**(Type) - Description**
limt	A LimitSegment **(object)**
index	**(variant)** - Limit line number in the LimitTest collection
value	**(enum NALimitSegmentType)** - Limit Line type. Choose from: **0 - naLimitSegmentType_OFF** - turns limit line OFF **1 - naLimitSegmentType_Maximum** - limit line fails with a data point ABOVE the line **2 - naLimitSegmentType_Minimum** - limit line fails with a data point BELOW the line
Return Type	Long Integer
Default	0 - OFF
Examples	Set limts = meas.LimitTest limts.Type = naLimitSegmentType_Maximum 'Write limitType = limts.Type 'Read
C++ Syntax	HRESULT put_Type(tagNALimitSegmentType *pVal) HRESULT get_Type(tagNALimitSegmentType newVal)
Interface	ILimitSegment

Marker Object

Marker Object (default interface is IMarker)

Description

Contains the methods and properties that control Markers.

To turn ON a marker, get a handle to the marker through the measurement object. (There is no markers collection).

If not already activated, this command will turn ON marker 1

```
Measurement.marker(1).Format = naLinMag
```

You can also set the marker object to an object variable:

```
Dim m1 As Marker
```

```
Set m1 = meas.marker(1)
```

There are 10 markers available per measurement:

- 1 reference marker
- 9 markers for absolute data or data relative to the reference marker (delta markers).

There are two ways to control markers through COM.

1. The Measurement object has properties that apply to all of the markers for that measurement.
2. Marker object properties override the Measurement object properties. For example, **meas.MarkerFormat = naLinMag** applies formatting to all markers. You can then override that setting for an individual marker by specifying **mark.Format = naLogMag** on the marker object.

Note: SearchFilterBandwidth is available through the measurement object.

Methods	**Description**
Activate	Makes an object the Active Object.
	Shared with the Marker Object
SearchMax	Searches the marker domain for the maximum value.
SearchMin	Searches the marker domain for the minimum value.
SearchNextPeak	Searches the marker's domain for the next largest peak value.
SearchPeakLeft	Searches the marker's domain for the next VALID peak to the left of the marker.
SearchPeakRight	Searches the marker's domain for the next VALID peak to the right of the marker.
SearchTarget	Searches the marker's domain for the target value.
SearchTargetLeft	Moving to the left of the marker position, searches the marker's domain for the target value.
SearchTargetRight	Moving to the right of the marker position, searches the marker's domain for the target value.
SetCenter	Changes the analyzer's center frequency to the X-axis position of the marker.
SetCW	Changes the analyzer to sweep type CW mode and makes the CW frequency the marker's frequency.
SetElectricalDelay	Changes the measurement's electrical delay to the marker's delay value.
SetReferenceLevel	Changes the measurement's reference level to the marker's Y-axis value.
SetStart	Changes the analyzer's start frequency to the X-axis position of the marker.
SetStop	Changes the analyzer's stop frequency to the X-axis position of the marker.
Property	**Description**
Bucket Number	Marker data point number
DeltaMarker	Makes a marker relative to the reference marker
Format	Linear, SWR, and so forth
Interpolated	Turn marker interpolation ON and OFF
Number	Read the number of the active marker
PeakExcursion	Sets and reads the peak excursion value for the specified marker.
PeakThreshold	Sets peak threshold for the specified marker.
SearchFunction	Emulates the Tracking function in the marker search dialog box.
Stimulus	Sets and reads the X-Axis value of the marker.
Target Value	Sets the target value for the marker when doing Target Searches.
Tracking	The tracking function finds the selected search function every sweep.
Type	Sets and reads the marker type.
UserRange	Assigns the marker to the specified User Range.
UserRangeMax	Sets the stimulus stop value for the specified User Range.

UserRangeMin	Sets the stimulus start value for the specified User Range.
Value	Reads the Y-Axis value of the marker.

Write-only

Activate Method

Description	Makes an object the Active Object. When making a measurement active, the channel and window the measurement is contained in becomes the active channel and active window. In order to change properties on any of the active objects, you must first have a "handle" to the active object using the **Set** command. For more information, See Getting a Handle to an Object. You do not have to make an object "Active" to set or read its properties remotely. But an object must be "Active" to change its values from the front panel.
VB Syntax	*object.***Activate**
Variable	**(Type) - Description**
object	Measurement **(object)** **or** Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.Activate mark.Activate
C++ Syntax	HRESULT Activate()
Interface	IMeasurement IMarker

Write-only **About Marker Search**

SearchMax Method

Description	Searches the marker domain for the maximum value.
VB Syntax	*mark.***SearchMax**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchMax
C++ Syntax	HRESULT SearchMax()
Interface	IMarker

Write-only **About Marker Search**

SearchMin Method

Description	Searches the marker domain for the minimum value.
VB Syntax	*mark*.**SearchMin**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchMin
C++ Syntax	HRESULT SearchMin()
Interface	IMarker

Write-only **About Marker Search**

SearchNextPeak Method

Description	Searches the marker's domain for the next peak value.
VB Syntax	*mark*.**SearchNextPeak**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchNextPeak
C++ Syntax	HRESULT SearchNextPeak()
Interface	IMarker

Write-only **About Marker Search**

SearchPeakLeft Method

Description	Searches the marker's domain for the next **VALID** peak to the left of the marker.
VB Syntax	*mark*.**SearchPeakLeft**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchPeakLeft
C++ Syntax	HRESULT SearchPeakLeft()
Interface	IMarker

Write-only **About Marker Search**

SearchPeakRight Method

Description	Searches the marker's domain for the next **VALID** peak to the right of the marker.
VB Syntax	*mark*.**SearchPeakRight**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchPeakRight
C++ Syntax	HRESULT SearchPeakRight()
Interface	IMarker

Write-only **About Marker Search**

SearchTarget Method

Description	Searches the marker's domain for the target value (specified with mark.TargetValue). Searches to the right; then at the end of the search domain, begins again at the start of the search domain.
VB Syntax	*mark*.**SearchTarget**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchTarget
C++ Syntax	HRESULT SearchTarget()
Interface	IMarker

Write-only **About Marker Search**

SearchTargetLeft Method

Description	Moving to the left of the marker position, searches the marker's domain for the target value (specified with mark.TargetValue).
VB Syntax	*mark*.**SearchTargetLeft**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchTargetLeft
C++ Syntax	HRESULT SearchTargetLeft()

Interface IMarker

Write-only **About Marker Search**

SearchTargetRight Method

Description	Moving to the right of the marker position, searches the marker's domain for the target value (specified with mark.TargetValue).
VB Syntax	*mark.***SearchTargetRight**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SearchTargetRight
C++ Syntax	HRESULT SearchTargetRight()
Interface	IMarker

Write-only **About Marker Functions**

SetCenter Method

Description	Changes the center stimulus to the stimulus value of the marker. The start stimulus stays the same and the stop is adjusted.
VB Syntax	*mark.***SetCenter**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SetCenter
C++ Syntax	HRESULT SetCenter()
Interface	IMarker

Write-only **About Marker Functions**

SetCW Method

Description	Changes the analyzer to sweep type CW mode and sets the CW frequency to the marker's frequency. Does not change anything if current sweep type is other than a frequency sweep.
VB Syntax	*mark.***SetCW**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples	mark.SetCW
C++ Syntax	HRESULT SetCW()
Interface	IMarker

Write-only **About Marker Functions**

SetElectricalDelay Method

Description	Changes the measurement's electrical delay to the marker's delay value.
VB Syntax	*mark*.**SetElectricalDelay**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SetElectricalDelay
C++ Syntax	HRESULT SetElectricalDelay()
Interface	IMarker

Write-only **About Marker Functions**

SetReferenceLevel Method

Description	Changes the measurement's reference level to the marker's Y-axis value.
VB Syntax	*mark*.**SetReferenceLevel**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	mark.SetReferenceLevel
C++ Syntax	HRESULT SetReferenceLevel()
Interface	IMarker

Write-only **About Marker Functions**

SetStart Method

Description	Changes the start stimulus to the stimulus value of the marker. The stop stimulus stays the same and the span is adjusted.
VB Syntax	*mark*.**SetStart**
Variable	**(Type) - Description**
mark	A Marker **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples mark.SetStart

C++ Syntax HRESULT SetStart()
Interface IMarker

Write-only **About Marker Functions**

SetStop Method

Description Changes the stop stimulus to the stimulus value of the marker. The start stimulus stays the same and the span is adjusted.
VB Syntax *mark*.**SetStop**

Variable **(Type) - Description**
mark A Marker **(object)**
Return Type Not Applicable
Default Not Applicable

Examples mark.SetStop

C++ Syntax HRESULT SetStop()
Interface IMarker

Write/Read **About Markers**

BucketNumber Property

Description Sets or returns the bucket number (data point) for the active marker.
VB Syntax *mark*.**BucketNumber** = *value*

Variable **(Type) - Description**
mark A Marker **(object)**
value **(long integer)** - Data point. Choose any number between 0 and the measurement's number of data points - 1.
For example, with Number of points = 201, choose between 0 and 200
Return Type Long Integer
Default The first marker is set to the middle of the span. Subsequent markers are set to the bucket number of the previously active marker.

Examples mark.BucketNumber = 100 'moves the active marker to data point 100 - Write
pointNumber = mark.BucketNumber 'returns the data point number the active marker is currently on. -Read

C++ Syntax HRESULT get_BucketNumber(long *pVal)
HRESULT put_BucketNumber(long newVal)
Interface IMarker

Write/Read **About Reference Markers**

DeltaMarker Property

Description	Sets a marker as a delta marker. The reference marker must already be turned ON. See meas.ReferenceMarkerState
VB Syntax	*mark*.**DeltaMarker** = *state*
Variable	**(Type) - Description**
app	A Marker **(object)**
state	**(boolean)** - ON (1) marker is a delta marker OFF (0) marker is NOT a delta marker
Return Type	Boolean
Default	OFF (0)
Examples	mark.DeltaMarker = True 'Write delta = mark.DeltaMarker 'Read
C++ Syntax	HRESULT get_DeltaMarker(VARIANT_BOOL bState) HRESULT put_DeltaMarker(VARIANT_BOOL *bState)
Interface	IMarker

Write/Read **About Marker Format**

Format Property (marker)

Description	Sets (or returns) the format of the marker.
VB Syntax	*mark*.**Format** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(enum NAMarkerFormat)** - Choose from: **0** - naMarkerFormat_LinMag **1** - naMarkerFormat_LogMag **2** - naMarkerFormat_Phase **3** - naMarkerFormat_Delay **4** - naMarkerFormat_Real **5** - naMarkerFormat_Imaginary **6** - naMarkerFormat_SWR **7** - naMarkerFormat_LinMagPhase **8** - naMarkerFormat_LogMagPhase **9** - naMarkerFormat_RealImaginary **10** - naMarkerFormat_ComplexImpedance **11** - naMarkerFormat_ComplexAdmittance
Return Type	NAMarkerFormat
Default	**1** - naMarkerFormat_LogMag
Examples	mark.Format = naMarkerFormat_SWR 'Write fmt = mark.Format 'Read
C++ Syntax	HRESULT get_Format(tagNAMarkerFormat *pVal) HRESULT put_Format(tagNAMarkerFormat newVal)
Interface	IMarker

Write/Read **About Markers**

Interpolated Property

Description	Turns marker Interpolation ON and OFF. Marker interpolation enables X-axis resolution beyond the discrete data values. The analyzer will calculate the x and y-axis data values between discrete data points. Use meas.Interpolate to change interpolation of **all** markers in a measurement. This command will override the measurement setting.
VB Syntax	*mark*.**Interpolated** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(boolean)** **False** - Turns interpolation OFF **True** - Turns interpolation ON
Return Type	Boolean
Default	True (ON)
Examples	mark.Interpolated = 1 'Write interpolate = mark.Interpolated 'Read
C++ Syntax	HRESULT get_Interpolated(VARIANT_BOOL *pVal) HRESULT put_Interpolated(VARIANT_BOOL newVal)
Interface	IMarker

Read-only **About Markers**

Number Property

Description	Returns the number of the marker.
VB Syntax	*marknum = mark*.**Number**
Variable	**(Type) - Description**
marknum	**(long)** - Variable to store marker number
mark	A Marker **(object)**
Return Type	Long Integer
Default	Not applicable
Examples	marknum = mark.Number 'Read
C++ Syntax	HRESULT get_Number(long *pVal)
Interface	IMarker

Write/Read **About Marker Search**

PeakExcursion Property

Description	Sets and reads the peak excursion value for the specified marker. The Excursion value determines what is considered a "peak".
VB Syntax	*mark*.**PeakExcursion** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(single)** - Peak Excursion. Choose any number between **-500** and **500**
Return Type	Single
Default	3
Examples	mark.PeakExcursion = 1 'Write PkExcur = mark.PeakExcursion 'Read
C++ Syntax	HRESULT get_PeakExcursion(float *pVal) HRESULT put_PeakExcursion(float newVal)
Interface	IMarker

Write/Read **About Marker Search**

PeakThreshold Property

Description	Sets peak threshold for the specified marker. If a peak (using the criteria set with PeakExcursion) is below this reference value, it will not be considered when searching for peaks.
VB Syntax	*mark*.**PeakThreshold** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(single)** - Peak Threshold. Choose any number between: **-500** and **500**
Return Type	Single
Default	-100db
Examples	mark.PeakThreshold = 1 'Write PkThresh = mark.PeakThreshold 'Read
C++ Syntax	HRESULT get_PeakThreshold(float *pVal) HRESULT put_PeakThreshold(float newVal)
Interface	IMarker

Write/Read **About Marker Search**

SearchFunction Property

Description	Emulates the Tracking function in the marker search dialog box. The value you choose for SearchFunction will determine the type of search that takes place when the Tracking property is set true. The tracking function finds the selected search function every sweep. In

	effect, turning Tracking ON is the same as executing one of the "Search..." methods (such as SearchMin, SearchMax) for every sweep.
VB Syntax	*mark*.**SearchFunction** = *value*

Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(enum NAMarkerFunction)** - search function. Choose from: **0** - naMarkerFunction_**None** **1** - naMarkerFunction_**Min** **2** - naMarkerFunction_**Max** **3** - naMarkerFunction_**Target** **4** - naMarkerFunction_**NextPeak** **5** - naMarkerFunction_**PeakRight** **6** - naMarkerFunction_**PeakLeft**
Return Type	Long Integer
Default	**0** - naMarkerFunction_None

Examples	mark.SearchFunction = naMarkerFunction_Target 'When this marker is set to track, it will track the Target value. searchfunction = mark.SearchFunction 'Read

C++ Syntax	HRESULT get_SearchFunction(tagNAMarkerFunction *pVal) HRESULT put_SearchFunction(tagNAMarkerFunction newVal)
Interface	IMarker

Write/Read **About Markers**

Stimulus Property

Description	Sets and reads the X-Axis value of the marker. If the marker is a delta marker, the value will be relative to the reference marker.
VB Syntax	*mark*.**Stimulus** = *value*

Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(double)** - X-Axis value. Choose any number within the full span of the channel or User Range (if set).
Return Type	Double
Default	First activated Marker turns ON in the middle of the X-axis range. Subsequent markers turn ON at the position of the most recently active marker.

Examples	mark.Stimulus = 3e9 'Write XVal = mark.Stimulus 'Read

C++ Syntax	HRESULT get_Stimulus(double *pVal) HRESULT put_Stimulus(double newVal)
Interface	IMarker

Write/Read **About Marker Search**

TargetValue Property

Description	Sets the target value for the marker when doing Target Searches (SearchTargetLeft, SearchTarget, SearchTargetRight).
VB Syntax	*mark*.**TargetValue** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(single)** - Target value. Choose any number between: **-500** and **500**
Return Type	Single
Default	0
Examples	mark.TargetValue = 10.5 'Write target = mark.TargetValue 'Read
C++ Syntax	HRESULT get_TargetValue(float *pVal) HRESULT put_TargetValue(float newVal)
Interface	IMarker

Write/Read **About Marker Search**

Tracking Property

Description	This property, when on, executes the search function (marker.SearchFunction) every sweep. In effect, turning Tracking ON is the same as executing one of the immediate, one-time, "Search..." methods (such as SearchMin, SearchMax) for every sweep.
VB Syntax	*mark*.**Tracking** = *state*
Variable	**(Type) - Description**
mark	A Marker **(object)**
state	**(boolean)** - Tracking state. Choose from: **ON (1)** **OFF (0)**
Return Type	Boolean **0 -** Tracking OFF **1 -** Tracking ON
Default	0 - OFF
Examples	mark.Tracking = 1 'Write markTracking = mark.Type 'Read
C++ Syntax	HRESULT put_Tracking(VARIANT_BOOL bOn) HRESULT get_Tracking(VARIANT_BOOL * pbOn)
Interface	IMarker

Write/Read **About Marker Types**

Type (Marker) Property

Description	Sets and reads the marker type.
VB Syntax	*mark*.**Type** = *value*
Variable	**(Type) - Description**
chan	A Marker **(object)**
value	**(enum NAMarkerType)** - Marker Type. Choose from: **0 - naMarkerType_Normal -** the X-axis value for a normal marker will always be determined by the measurement data of the marker. **1 - naMarkerType_Fixed -** retains and keeps its x-axis value at the time the marker type is set.
Return Type	Long Integer
Default	naMarkerType_Normal
Examples	mark.Type = naMarkerType_Normal 'Write MrkType = mark.Type 'Read
C++ Syntax	HRESULT get_Type(tagNAMarkerType *pVal) HRESULT put_Type(tagNAMarkerType newVal)
Interface	IMarker

Write/Read **About User Ranges**

UserRange Property

Description

Assigns the marker to the specified User Range. This restricts the marker's x-axis travel to the User Range span, specified with Start and Stop values.

- Each channel has 10 user ranges.
- Markers and trace statistics can be restricted to any user range.
- More than one marker can occupy a user range.
- User ranges can overlap. For example:
  - User range 1 - 3GHz to 5GHz
  - User range 2 - 4GHz to 6GHz

Note: User ranges are especially useful in restricting marker searches to specific areas of the measurement.

VB Syntax	*mark*.**UserRange** = *value*
Variable	**(Type) - Description**
mark	A Marker **(object)**
value	**(long integer)** - User Range. Choose any number between: **0** and **9 (0=Full Span)**
Return Type	Long Integer
Default	0 - Full Span
Examples	mark.UserRange = 1 'Write UseRnge = mark.UserRange 'Read
C++ Syntax	HRESULT get_UserRange(long *pRangeNumber)

	HRESULT put_UserRange(long lRangeNumber)
Interface	IMarker

Read-only **About Markers**

Value Property

Description	Reads the Y-Axis value of the marker. If the marker is a delta marker, the value will be relative to the reference marker. You cannot set the Y-axis value of a marker. The marker remains at the position at the time you set marker.Type.
VB Syntax	*YValue* = *mark*.**Value** *(format)*
Variable	**(Type) - Description**
YValue	A variable to store the Y-axis value
mark	A Marker **(object)**
format	**(enum NAMarkerFormat)** - The format you would like the marker's Y-axis value. The number in parenthesis following the format is the number of values that are returned in a variant array. Choose from: 0 - naMarkerFormat_**LinMag** (1) 1 - naMarkerFormat_**LogMag** (1) 2 - naMarkerFormat_**Phase** (1) 3 - naMarkerFormat_**Delay** (1) 4 - naMarkerFormat_**Real** (1) 5 - naMarkerFormat_**Imaginary** (1) 6 - naMarkerFormat_**SWR** (1) 7 - naMarkerFormat_**LinMagPhase** (2) 8 - naMarkerFormat_**LogMagPhase** (2) 9 - naMarkerFormat_**RealImaginary** (2) 10 - naMarkerFormat_**ComplexImpedance** (3) 11 - naMarkerFormat_**ComplexAdmittance** (3)
Return Type	Variant - The previous list of formats indicates the number of values that are returned in a variant array
Default	Not applicable
Examples	YVal = mark.Value 'Read
C++ Syntax	HRESULT get_Value(tagNAMarkerFormat format, VARIANT *pVal)
Interface	IMarker

Measurements Collection

Measurement Collection

Description

A collection object that provides a mechanism for iterating through the Application measurements. See Collections in the Analyzer.

Methods	**Description**
Add	Adds a Measurement to the collection.
Item	Use to get a handle to a channel in the collection.
Remove	Removes a measurement from the measurements collection.
Properties	**Description**
Count	Returns the number of measurements in the analyzer.
Parent	Returns a handle to the current Application.

Write-only

Add (measurement) Method

Description Adds a Measurement to the collection.

VB Syntax *meas.***Add** *channel,param,source[,window]*

meas A Measurements collection **(object)**

channel **(long)** - Channel number of the new measurement.

param **(string)** - Parameter of the new measurement. Choose from:

- **"S11"**
- **"S22"**
- **"S21"**
- **"S12"**
- **"A"**
- **"B"**
- **"R1"**
- **"R2"**

or

combine 2 of (A,B,R1,R2) in this format: "A/R1"

source **(long integer)** - Source port number; if unspecified, value is set to 1. Only used for non-s-parameter measurements; ignored if s-parameter.

window **(long integer) -** Optional argument. Window number of the new measurement. Choose 1 to 4. If unspecified, the S-Parameter will be created in the Active Window.

Return Type None

Default None

Examples meass.Add 3,"A/R1",1,1 'Adds A/R1 measurement to channel 3 in window 1

C++ Syntax HRESULT Add(long ChannelNum, BSTR strParameter, long srcPort, VARIANT_BOOL bNewWindow)

Interface IMeasurements

Measurement Object

Measurement Object (default interface is IMeasurement2)

Description

The Measurement object is probably the most used object in the model. A measurement object represents the chain of data processing algorithms that take raw data from the channel and make it ready for display, which then becomes the scope of the Trace object.

A Measurement object is defined by it's parameter (S11, S22, A/R1, B and so forth). The measurement object is associated with a channel in that a channel drives the hardware that produces the data that feeds the measurement. The root of a measurement is the raw data. This buffer of complex paired data then flows through a number of processing blocks: error-correction, trace math, phase correction, time domain, gating, formatting. All of these are controlled through the measurement object.

The active measurement determines what ever else is active. The active measurement is best described as the measurement that will be acted upon if you make a setting from the front panel. It is the measurement whose "button" is pressed in the window with the red "active window" frame. If you create a new measurement, that measurement becomes the active measurement.

Therefore, all automation methods with the word "Active" in them refer to the object associated with the Active measurement, whether that object is a Channel, Window, Trace or Limit line.

You can access two other objects through the Measurement object: markers and limit test. For example, because each measurement has its own set of markers, you can set a marker by doing this:

```
Dim meas as measurement
 Set meas = pna.ActiveMeasurement
 Meas.marker(1).Stimulus = 900e6
 Meas.LimitTest.State = true ' on
```

The IMeasurement2 interface

This interface extends the IMeasurement Interface and is the **default interface**. Some of the properties and methods for the IMeasurement2 Interface return stimulus values.

Every measurement carries with it a snapshot of the stimulus properties of the channel that were in effect when the measurement last acquired data. Therefore, it is the measurement that provides the most accurate stimulus description of its data. Any change made to the channel after the measurement was acquired renders the IChannel interface unreliable in terms of describing the measurement.

See also Custom Interfaces

- **IArrayTransfer**
- **IMixer used with (Option 083)**

Methods	**Interface**	**Description**
Activate	IMeasurement	Makes an object the Active Object. Shared with the Marker Object
ActivateMarker	IMeasurement	Makes a marker the Active Marker.
ChangeParameter	IMeasurement	Changes the parameter of the measurement.
DataToDivisor	IMeasurement	Stores data for receiver power cal of unratioed measurements
DataToMemory	IMeasurement	Stores the active measurement into memory.

Delete	IMeasurement	Deletes the measurement object.
DeleteAllMarkers	IMeasurement	Deletes all of the markers from the measurement.
DeleteMarker	IMeasurement	Deletes a marker from the active measurement.
getData	IMeasurement	Retrieves Complex data from analyzer memory
getDataByString	IMeasurement	Retrieves variant data from the specified location in your choice of formats.
GetXAxisValues	**IMeasurement2**	Returns the stimulus values for the specified measurement.
GetFilterStatistics	IMeasurement	Returns all four Filter Statistics
GetReferenceMarker	IMeasurement	Returns a handle to the reference marker.
GetTraceStatistics	IMeasurement	Returns the Trace Statistics.
InterpolateMarkers	IMeasurement	Turns All Marker Interpolation ON and OFF for the measurement.
putDataComplex	IMeasurement	Puts complex data into one of five data buffers.
putDataScalar	IMeasurement	Puts formatted variant data into the measurement results buffer.
SearchFilterBandwidth	IMeasurement	Searches the domain with the current BW target.
Properties		**Description**
ActiveMarker	IMeasurement	Returns a handle to the Active Marker object.
BandwidthTarget	IMeasurement	The insertion loss value at which the bandwidth of a filter is measured.
BandwidthTracking	IMeasurement	Turns Bandwidth Tracking function ON and OFF.
CalibrationName	**IMeasurement2**	Get the name of the current cal type.
CalibrationType	IMeasurement	Set or get the calibration type for the measurement.
CalibrationTypeID	**IMeasurement2**	Sets or returns the current cal type to the Cal Set ID (CLSID).
Center	**IMeasurement2**	Returns the stimulus value of the center point for the measurement. This function does not work for segment sweep measurements.
channelNumber	IIMeasurement	Returns the channel number. Shared with the Channel Object
Domain	**IMeasurement2**	Returns the domain (frequency, time, power) for the measurement.
ElectricalDelay	IMeasurement	Sets electrical delay.
ErrorCorrection	IMeasurement	Set or get the state of error correction for the measurement.
FilterBW	IMeasurement	Returns the results of the SearchBandwidth method.
FilterCF	IMeasurement	Returns the Center Frequency result of the SearchBandwidth method.
FilterLoss	IMeasurement	Returns the Loss value of the SearchBandwidth method.
FilterQ	IMeasurement	Returns the Q (quality factor) result of the SearchBandwidth method.
Format	IMeasurement	Sets display format.
Gating (object)		
InterpolateCorrection	IMeasurement	Turns ON and OFF the calculation of new error terms when stimulus values change.
InterpolateNormalization	IMeasurement	Turns ON and OFF normalization interpolation when stimulus values change after receiver power cal of unratioed measurements.
IsSparameter	**IMeasurement2**	Returns true if measurement represents an S-Parameter.
LimitTest (collection)		
LimitTestFailed	IMeasurement	Returns the results of limit testing

LoadPort	IMeasurement	Returns the load port number associated with an S-parameter reflection measurement.
LogMagnitudeOffset	IMeasurement	Sets or returns the value that normalized, unratioed, receiver power measurement data will be shifted by
marker (object)		
MarkerFormat	IMeasurement	Sets or returns the format of all the markers in the measurement.
Mean	IMeasurement	Returns the mean value of the measurement.
Name	IMeasurement	Sets or returns the name of the measurement.
NAWindow (object)		
Normalization	IMeasurement	Turns ON or OFF normalization for receiver power cal of unratioed measurements
Number	IMeasurement	Returns the number of the measurement.
NumberOfPoints	**IMeasurement2**	Returns the Number of Points of the measurement.
Parameter	IMeasurement	Returns the measurement Parameter.
PeakToPeak	IMeasurement	Returns the Peak to Peak value of the measurement.
PhaseOffset	IMeasurement	Sets the Phase Offset for the active channel.
ReceivePort	**IMeasurement2**	Returns the receiver port of the measurement.
ReferenceMarkerState	IMeasurement	Turns the reference marker ON or OFF
ShowStatistics	IMeasurement	Displays and hides the measurement statistics (peak-to-peak, mean, standard deviation) on the screen.
Smoothing	IMeasurement	Turns ON and OFF data smoothing.
SmoothingAperture	IMeasurement	Specifies or returns the amount of smoothing as a ratio of the number of data points in the measurement trace.
SourcePort	**IMeasurement2**	Returns the source port of the measurement.
Span	**IMeasurement2**	Returns the stimulus span (stop - start) for the measurement.
StandardDeviation	IMeasurement	Returns the standard deviation of the measurement.
Start	**IMeasurement2**	Returns the stimulus value of the first point for the measurement.
StatisticsRange	IMeasurement	Sets the User Range number for calculating measurement statistics.
Stop	**IMeasurement2**	Returns the stimulus value of the last point for the measurement.
Trace (object)		
TraceMath	IMeasurement	Performs math operations on the measurement object and the trace stored in memory.
Transform (object)		
View	IMeasurement	Sets (or returns) the type of trace displayed on the screen

IArrayTransfer Interface

Description

Contains methods for putting data in and getting data out of the analyzer using typed data. This interface transfers data more efficiently than the default IMeasurement Interface.

Method	Description
getComplex	Retrieves real and imaginary data from the specified buffer.
getNAComplex	Retrieves typed **NAComplex** data from the specified buffer.
getPairedData	Retrieves magnitude and phase data pairs from the specified buffer.
getScalar	Retrieves scalar data from the specified buffer.
putComplex	Puts real and imaginary data into the specified buffer.
putNAComplex	Puts typed **NAComplex** data into the specified buffer.
putScalar	Puts scalar data into the measurement result buffer.
Property	**Description**
None	

MIXER Interface used with (Option 083)

Methods	Interface	Description
LoadFile	**IMixer**	Loads a previously-configured mixer attributes file (**.mxr**)
SaveFile	**IMixer**	Saves the settings for the mixer/converter test setup to a mixer attributes file.
Properties	**Interface**	**Description**
ElecDelayMedium	**IMixer**	Sets or returns the characteristic of the electrical delay medium.
IFDenominator	**IMixer**	Sets or returns the denominator value of the IF Fractional Multiplier.
IFNumerator	**IMixer**	Sets or returns the numerator value of the IF Fractional Multiplier.
IFSideband	**IMixer**	Sets or returns the value of the IF sideband.
IFStartFrequency	**IMixer**	Returns the start frequency value of the mixer IF frequency.
IFStopFrequency	**IMixer**	Returns the stop frequency value of the mixer IF frequency.
InputDenominator	**IMixer**	Sets or returns the denominator value of the Input Fractional Multiplier.
InputNumerator	**IMixer**	Sets or returns the numerator value of the Input Fractional Multiplier.
InputPower	**IMixer**	Sets or returns the value of the Input Power.
InputStartFrequency	**IMixer**	Sets or returns the start frequency value of the mixer input frequency.
InputStopFrequency	**IMixer**	Sets or returns the stop frequency value of the mixer input frequency.
LODenominator	**IMixer**	Sets or returns the denominator value of the LO Fractional Multiplier.
LOFixedFrequency	**IMixer**	Sets or returns the fixed value of the LO frequency.
FixedFrequency	**IMixer**	Sets or returns the frequency fixed value.
LOName	**IMixer**	Sets or returns the LO name.
LONumerator	**IMixer**	Sets or returns the numerator value of the LO Fractional Multiplier.
LOPower	**IMixer**	Sets or returns the value of the LO Power
LOStage	**IMIXer**	Returns the number of stages.
OutputDenominator	**IMIXer**	Sets or returns the denominator value of the output Fractional Multiplier.
OutputNumerator	**IMIXer**	Sets or returns the numerator value of the output Fractional Multiplier.
OutputSideban	**IMIXer**	Sets or returns the value of the output sideband.

d		
OutputStartFrequency	**IMIXer**	Sets or returns the start frequency value of the mixer output frequency.
OutputStopFrequency	**IMIXer**	Sets or returns the stop frequency value of the mixer output frequency
WGCutoffFreq	**IMIXer**	Sets or returns the value of the waveguide cut off frequency

Write-only **About Markers**

ActivateMarker Method

Description	Makes a marker the Active Marker. Use meas.ActiveMarker to read the number of the active marker.
VB Syntax	*meas.*__ActivateMarker(__*Mnum*__)__
Variable	**(Type) - Description**
meas	A Measurement **(object)**
Mnum	**(long integer)** - the number of the marker to make active. Choose any marker number from **1** to **9**.
Return Type	None
Default	Not Applicable
Examples	meas.ActivateMarker(1)'Write
C++ Syntax	HRESULT ActivateMarker(long lMarkerNumber)
Interface	IMeasurement
Remarks	Use ReferenceMarkerState to control the Reference marker.

Read only

CalibrationName Property

Description	Get the name of the current cal type.
VB Syntax	*value = meas.*__CalibrationName__
Variable	**(Type) - Description**
value	**(string)** - Variable to store the returned value.
meas	A Measurement **(object)**
Return Type	String
Default	Not Applicable
Examples	ct = meas.CalibrationName
C++ Syntax	HRESULT get_CalibrationName([out,retval] BSTR* CalibrationName);
Interface	IMeasurement2

Write/Read

CalibrationTypeID Property

Description	Sets or returns the current cal type measurements using a CLSID (or GUID) associated with the caltype.
VB Syntax	*meas*.**CalibrationTypeID**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
Return Type	String
Default	Not Applicable

Examples

```
Dim pna
Dim m

  Const VMCCalTypeID = "{2061767B-0FE2-4F6F-86D0-9AB332B18DA5}"
  Const VMCCalTypeProgID = "VectorMixerCal.VCMCType"
  Set pna = CreateObject("AgilentPNA835x.Application")
  Set m = pna.ActiveMeasurement
                        ' set the type using the GUID
  m.CalibrationTypeID = VMCCalTypeID
                        ' or set the type using the PROGID
  m.CalibrationTypeID = VMCCalTypeProgID
  m.ErrorCorrection = True
MsgBox   m.CalibrationName
```

C++ Syntax	HRESULT get_CalibrationTypeID([out, retval] BSTR* CalibrationTypeID); HRESULT put_CalibrationTypeID([in] BSTR CalibrationTypeID);
Interface	IMeasurement2

Write-only **About Measurement Parameters**

ChangeParameter Method

Description	Changes the parameter of the measurement.
VB Syntax	*meas*.**ChangeParameter**(*param,lPort*)
Variable	**(Type) - Description**
meas	A Measurement **(object)**
param	**(string)** - New parameter. Choose from: **S11 I S22 I S21 I S12** Additionally, for 3-port analyzers only: **S33 I S13 I S31 I S23 I S32** For non-ratioed measurements: **A I B I R1 I R2** **C -** 3-port analyzers only For ratioed measurements: **A/B** **A/C** - 3 port analyzers only

B/A

B/C - 3 port analyzers only

C/A - 3 port analyzers only

C/B - 3 port analyzers only

A/R1

B/R1

C/R1 - 3 port analyzers only

A/R2

B/R2

R1/A

R2/A

R1/B

R2/B

R1/C - 3 port analyzers only

R2/R1

R1/R2

lPort	**(long integer)** Load port if *param* is a reflection S-Parameter Ignored if *param* is a transmission S-Parameter Source port if *param* is anything other than an S-parameter
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.ChangeParameter "S11",1
C++ Syntax	HRESULT ChangeParameter(BSTR parameter, long lPort)
Interface	IMeasurement

Write-only **About Receiver Cal**

DataToDivisor Method

Description	Stores the measurement's data to the measurement's "divisor" buffer for use by the Normalization data processing algorithm. Normalization is currently supported only on measurements of unratioed power, for purpose of receiver power calibration. If DataToDivisor is called on a ratioed measurement (such as an S-parameter), it will return an error.
VB Syntax	*meas.***DataToDivisor**
Variable	**(Type) - Description**
meas	**(object) -** A Measurement object
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.DataToDivisor
C++ Syntax	HRESULT DataToDivisor();
Interface	IMeasurement

Write-only **About Math Operations**

DataToMemory Method

Description	Stores the active measurement data into memory creating a memory trace. The memory can then be displayed or used in calculations with the measurement data.
VB Syntax	*meas.***DataToMemory**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.DataToMemory
C++ Syntax	HRESULT DataToMemory()
Interface	IMeasurement

Write-only **About Measurement Parameters**

Delete Method

Description	Deletes the measurement.
VB Syntax	*meas.***Delete**
Variable	**(Type) - Description**
meas	The Measurement object to delete **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.Delete

C++ Syntax	HRESULT Delete()
Interface	IMeasurement

Write-only **About Markers**

DeleteAllMarkers Method

Description	Deletes all of the markers from the measurement.
VB Syntax	*meas.***DeleteAllMarkers**
Variable	**(Type) - Description**
meas	**(object) -** The Measurement object from which markers will be deleted.
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.DeleteAllMarkers
C++ Syntax	HRESULT DeleteAllMarkers()
Interface	IMeasurement

Write-only **About Markers**

DeleteMarker Method

Description	Deletes a marker from the measurement.
VB Syntax	*meas.***DeleteMarker(***Mnum***)**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
Mnum	**(long)** - Any existing marker number in the measurement
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.DeleteMarker(1)
C++ Syntax	HRESULT DeleteMarker(long lMarkerNumber)
Interface	IMeasurement

Read-only **About Accessing Data**

GetData Method

Description	Retrieves variant data from the specified location in your choice of formats. **Note:** This method returns a variant which is less efficient than methods available on the IArrayTransfer interface. **Note:** If you plan to **Put** this data back into analyzer, putDataComplex (variant data) method requires complex, two-dimensional data. Therefore, request the data in **Polar** format.
VB Syntax	*data = meas.***GetData** *location, format*

Variable	(Type) - Description
data	Variant array to store the data.
meas	A Measurement **(object)**
location	**(enum NADataStore)** - Where the data you want is residing. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
format	**(enum NADataFormat)** - Format in which you would like the data. It does not have to be the displayed format. Choose from: • naDataFormat_LinMag • naDataFormat_LogMag • naDataFormat_Phase • **naDataFormat_Polar *** • **naDataFormat_Smith *** • naDataFormat_Delay • naDataFormat_Real • naDataFormat_Imaginary • naDataFormat_SWR *** Specfiy Smith or Polar** formats to obtain complex data pairs, which require a two-dimensional array **varData (numpts, 2)** to accomodate both real and imaginary data. All scalar formats return a single dimension **varData(numpts).**
Return Type	Variant array - automatically dimensioned to the size of the data
Default	Not Applicable

Examples

```
Dim varData As Variant
 varData = meas.GetData(naMeasResult,naDataFormat_Phase)
 'Print Data
 For i = 0 to chan.NumberOfPoints-1
  Print varData(i)
 Next i
```

C++ Syntax	HRESULT getData(tagNADataStore DataStore, tagDataFormat DataFormat, VARIANT *pData)
Interface	IMeasurement

Read-only **About Accessing Data**

getDataByString Method

Description	Retrieves variant data from the specified location in your choice of formats.
VB Syntax	*data = meas*.**getDataByString** *location, format*

Variable	**(Type) - Description**
data	**(variant)** - Array to store the data.
meas	**(object) -** A Measurement object
location	**(string)** – Name of the buffer to be read.
format	**(enum NADataFormat)** - Format in which you would like the data. It does not have to be the displayed format. Choose from:

- naDataFormat_LinMag
- naDataFormat_LogMag
- naDataFormat_Phase
- **naDataFormat_Polar ***
- **naDataFormat_Smith ***
- naDataFormat_Delay
- naDataFormat_Real
- naDataFormat_Imaginary
- naDataFormat_SWR

*** Specfiy Smith or Polar** formats to obtain complex data pairs, which require a two-dimensional array **varData (numpts, 2)** to accomodate both real and imaginary data.

All scalar formats return a single dimension **varData(numpts).**

Return Type	Variant array
Default	Not Applicable
Examples	meas.getDataByString “VectorResult0”, naDataFormat_Phase
C++ Syntax	HRESULT getDataByString(BSTR location, tagDataFormat dataFormat, VARIANT * pData);
Interface	IMeasurement

Read-only **About Marker Search**

GetFilterStatistics Method

Description	Returns all four Filter Statistics resulting from a SearchFilterBandwidth. To retrieve individual filter statistics, use meas.FilterCF, meas.FilterBW, meas.FilterLoss, meas.FilterQ properties.
VB Syntax	*meas.***GetFilterStatistics** *cf,bw,loss,q*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
cf,bw,loss,q	Dimensioned variables to store the returned values
Return Type	**(double)** *cf* **(single)** *bw,loss,q*
Default	Not Applicable
Examples	'Dimension variables Dim cf as Double Dim bw as Single Dim loss as Single Dim q as Single

meas.GetFileterStatistics cf,bw,loss,q

C++ Syntax	HRESULT GetFilterStatistics(double* centerFreq, float* bw, float* loss, float* quality)
Interface	IMeasurement

Write/Read **About Reference Markers**

GetReferenceMarker Method

Description	Returns a handle to the reference marker.
VB Syntax	*meas*.**GetReferenceMarker**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
Return Type	Object
Default	Not Applicable
Examples	meas.GetReferenceMarker
C++ Syntax	HRESULT GetReferenceMarker(IMarker** refMarker)
Interface	IMeasurement

Read-only **About Trace Statistics**

GetTraceStatistics Method

Description	Returns all four Trace Statistics. To retreive individual Trace statistics, use Mean, PeakToPeak, StandardDeviation properties. Use ShowStatistics to display the statistics of the screen.
VB Syntax	*meas*.**GetTraceStatistics** *pp,mean,stdev*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
pp,mean,stdev	**(double)** - Dimensioned variables to store the returned values
Return Type	Double
Default	Not Applicable

Examples

```
'Dimension variables
 Dim pp As Double
 Dim mean As Double
 Dim stdv As Double
 meas.GetTraceStatistics pp, mean, stdv
```

C++ Syntax	HRESULT GetTraceStatistics(double* pp, double* mean, double* stdDeviation)
Interface	IMeasurement

Read-only

GetXAxisValues

Description	Returns the stimulus values for the measurement. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*data = meas*.**GetXAxisValues**

Variable	**(Type) - Description**
data	**(Variant)** Array to store the data.
meas	A Measurement **(object)**
Return Type	Variant
Default	Not Applicable

Examples

```
Dim varData As Variant
 Dim i As Integer
 varData = meas.GetXAxisValues
 'Print Data
 For i = 0 To meas.NumberOfPoints - 1
 Print varData(i)
 Next i
```

C++ Syntax	HRESULT GetXAxisValues(VARIANT* xData);
Interface	IMeasurement2

Write-only **About Markers**

InterpolateMarkers Method

Description	Turns **All** Marker Interpolation ON and OFF for the measurement. Marker interpolation enables X-axis resolution between the discrete data values. The analyzer will calculate the x and y-axis data values between discrete data points. To override this property for individual markers, use the Interpolated property.
VB Syntax	*meas*.**Interpolate** = *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(boolean)** **False** - Turns interpolation OFF for all markers in the measurement **True** - Turns interpolation ON for all markers in the measurement
Return Type	Boolean
Default	True (ON)

Examples meas.Interpolate = 1

C++ Syntax	HRESULT InterpolateMarkers(VARIANT_BOOL bNewVal)
Interface	IMeasurement

Write

put_CalibrationTypeID Property

Description	Gives the current cal type to the Cal Set ID (CLSID).
VB Syntax	*meas*.**put_CalibrationTypeID =** value

Variable	**(Type) - Description**
value	**(string)** - Variable containing the current cal type .

meas	A Measurement **(object)**
Return Type	string
Default	Not Applicable
Examples	meas.put_CalibrationTypeID = ct
C++ Syntax	HRESULT put_CalibrationTypeID([in] BSTR CLSIDCalType);
Interface	IMeasurement2

Write-only **About Accessing Data**

PutDataComplex Method

Description

Puts complex data into the specified location. This method forces the channel into Hold mode to prevent the input data from being overwritten.

Data put in naRawData *(location*) will be re-processed whenever a change is made to the measurement attributes such as format or correction.

Data put in naMeasurement *(location*) will be overwritten by any measurement attribute changes.

VB Syntax *meas*.**putDataComplex** *location, data*

Variable	**(Type) - Description**
meas	A measurement **(object)**
location	**(enum NADataStore)** Where the Data will be put. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
data	**(variant)** - A **two-dimensional** variant array. **Note:** All buffers except naMeasResult and naMemoryResult require Complex data
Return Type	Not Applicable
Default	Not Applicable
Examples	meas.putDataComplex naMeasResult, varData
C++ Syntax	HRESULT putDataComplex(tagNADataStore DataStore, VARIANT complexData)
Interface	IMeasurement

Write-only **About Accessing Data**

PutDataScalar Method

Description

Puts formatted variant scalar data into the measurement result buffer. The data will be immediately processed and displayed. Subsequent changes to the measurement state will be reflected on the display.

VB Syntax *meas*.**putDataScalar** *format, data*

Variable	(Type) - Description
meas	A measurement **(object)**
format	**(enum NADataFormat)** Format of the data. Choose from: 1 - naDataFormat_LinMag 2 - naDataFormat_LogMag 3 - naDataFormat_Phase **4 - naDataFormat_Polar *** **5 - naDataFormat_Smith *** 6 - naDataFormat_Delay 7 - naDataFormat_Real 8 - naDataFormat_Imaginary 9 - naDataFormat_SWR *** Smith and Polar** formats require a two-dimensional array **varData (numpts, 2)** to accomodate both real and imaginary data. All other formats are a single dimension **varData(numpts).**
data	**(variant)** - A **two-dimensional** complex variant data array. **Note:** The getData (variant) method includes a "format" argument, which allows scalar (one-dimensional) data. To put data back into the "raw" data buffer using this (putDataComplex) method, specify **Polar** format when using the getData method.

Return Type Not Applicable

Default Not Applicable

Examples measData.putDataScalar naDataFormat_Real, varData

C++ Syntax HRESULT putDataScalar(tagNADataStore DataStore, VARIANT complexScalar)

Interface IMeasurement

Write-only **About Marker Search**

SearchFilterBandwidth Method

Description

Searches the measurement data with the current BandwidthTarget (default is -3). To continually track the filter bandwidth, use BandwidthTracking.

This feature uses markers 1-4. If not already, they are activated. To turn off these markers, either turn them off individually or DeleteAllMarkers.

The bandwidth statistics are displayed on the analyzer screen. To get the bandwidth statistics, use either GetFilterStatistics or FilterBW, FilterCF , FilterLoss ,or FilterQ.

The analyzer screen will show either Bandwidth statistics OR Trace statistics; not both.

To search a UserRange with the bandwidth search, first activate marker 1 and set the desired UserRange. Then send the SearchFilterBandwidth command. The user range used with bandwidth search only applies to marker 1 searching for the max value. The other markers may fall outside the user range.

VB Syntax	*meas.***SearchFilterBandwidth**

Variable	**(Type) - Description**
meas	A Measurement **(object)**
Return Type	Not Applicable
Default	Not Applicable

Examples	meas.SearchFilterBandwidth

C++ Syntax	HRESULT SearchFilterBandwidth()
Interface	IMeasurement

Read-only **About Markers**

ActiveMarker Property

Description	Returns a handle to the Active Marker object. You can either **(1)** use the handle directly to access Marker properties and methods, or **(2)** set a variable to the Marker object. The variable retains a handle to the original object if another Marker becomes active.
VB Syntax	1) *meas.***ActiveMarker.***<setting>* **or** 2) Set *mark* = *meas.***ActiveMarker**

Variable	**(Type) - Description**
meas	**(object)** - An Measurement object
<setting>	A marker property (or method) and arguments
mark	**(object)** - A marker object
Return Type	marker object
Default	None

Examples	Public mark as marker Set mark = meas.ActiveMarker

C++ Syntax	HRESULT get_ActiveMarker(IMarker** marker)
Interface	IMeasurement

Write/Read **About Marker Search**

BandwidthTarget Property

Description	Sets the insertion loss value at which the bandwidth of a filter is measured (using BandwidthTracking or SearchFilterBandwidth). For example, if you want to determine the filter bandwidth 3 db below the bandpass peak value, set BandwidthTarget to **-3**.
VB Syntax	*meas.***BandwidthTarget** = *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(single)** - Target value. Choose any number between **-500** and **500**

Return Type	Single
Default	-3
Examples	meas.BandwidthTarget = -3 'Write fbw = meas.BandwidthTarget 'Read
C++ Syntax	HRESULT put_BandwidthTarget(float target) HRESULT get_BandwidthTarget(float* target)
Interface	IMeasurement

Write/Read **About Marker Search**

BandwidthTracking Property

Descriaption

Searches continually (every sweep) for the current BandwidthTarget (default is -3). To search the filter bandwidth for ONE SWEEP only (not continually), use meas.SearchFilterBandwidth.

This feature uses markers 1-4. To turn off these markers, either turn them off individually or DeleteAllMarkers.

The bandwidth statistics are displayed on the analyzer screen. To get the bandwidth statistics, use either GetFilterStatistics or FilterBW, FilterCF , FilterLoss ,or FilterQ.

The analyzer screen will show either Bandwidth statistics OR Trace statistics; not both.

To restrict the search to a UserRange with the bandwidth search, first activate marker 1 and set the desired UserRange. Then send the SearchFilterBandwidth command. The user range used with bandwidth search only applies to marker 1 searching for the max value. The other markers may fall outside the user range.

VB Syntax	*meas*.**BandwidthTracking** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(boolean)** **1** - Turns bandwidth tracking ON **0** - Turns bandwidth tracking OFF
Return Type	Boolean
Default	0 - OFF
Examples	meas.BandwidthTracking = 1 'Write bwtrack = meas.BandwidthTracking 'Read
C++ Syntax	HRESULT put_BandwidthTracking(VARIANT_BOOL state) HRESULT get_BandwidthTracking(VARIANT_BOOL* state)
Interface	IMeasurement

Write/Read **About Performing a Calibration**

CalibrationType Property

Description	Specifies the type of calibration to perform or apply to the measurement.
VB Syntax	*meas*.**CalibrationType** = *type*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
type	**(enum NACalType)** - Calibration type. Choose from: **0** - naCalType_Response_Open **1** - naCalType_Response_Short **2** - naCalType_Response_Thru **3** - naCalType_Response_Thru_And_Isol **4** - naCalType_OnePort **5** - naCalType_TwoPort_SOLT **6** - naCalType_TwoPort_TRL **7 -** naCalType_None **8** - naCalType_ThreePort_SOLT
Return Type	**NACalType**
Default	naCalType_None
Examples	meas.CalibrationType = naCalType_Response_Open 'Write meascal = meas.CalibrationType 'Read
C++ Syntax	HRESULT put_CalibrationType (tagNACalType CalType) HRESULT get_CalibrationType (tagNACalType* pCalType)
Interface	IMeasurement

Read-only

Center Property

Description	Returns the stimulus value of the center data point for the measurement. This function does NOT work for segment sweep measurements. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value* = *meas*.**Center**
Variable	**(Type) - Description**
value	**(Double)** - Variable to store the returned value.
meas	A Measurement **(object)**
Return Type	Double
Default	Not Applicable
Examples	Print meas.Center 'prints the center data point
C++ Syntax	HRESULT get_Center(double * Val);
Interface	IMeasurement2

Read-only

Domain Property

Description	Returns the domain (frequency,time, power) of the measurement. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value* = *meas*.**Domain**
Variable	**(Type) - Description**
value	(**Enum** as NADomainType) - variable to store the returned value 0 - Frequency 1 - Time 2 - Power
meas	A Measurement **(object)**
Return Type	Enum as NADomainType
Default	Not Applicable
Examples	Print meas.Domain 'prints the value of the domain enum
C++ Syntax	HRESULT get_Domain(tagNADomainType * Val);
Interface	IMeasurement2

Write/Read **About Electrical Delay**

ElectricalDelay Property

Description	Sets the Electrical Delay for the active channel.
VB Syntax	*meas*.**ElectricalDelay** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Electrical Delay in seconds. Choose any number between **-9.99** and **9.99**
Return Type	Double
Default	0
Examples	meas.ElectricalDelay = 1e-3 'Write edelay = meas.ElectricalDelay 'Read
C++ Syntax	HRESULT get_ElectricalDelay(double *pVal) HRESULT put_ElectricalDelay(double newVal)
Interface	IMeasurement

Write/Read **About Performing a Calibration**

ErrorCorrection Property

Description	Sets (or returns) error correction ON or OFF for the measurement.
VB Syntax	*meas*.**ErrorCorrection** = *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(boolean)** **0** - Turns error correction OFF **1** - Turns error correction ON
Return Type	Boolean
Default	Not Applicable
Examples	meas.ErrorCorrection = 1 'Write errcorr = meas.ErrorCorrection 'Read
C++ Syntax	HRESULT put_ErrorCorrection (VARIANT_BOOL bState) HRESULT get_ErrorCorrection (VARIANT_BOOL *bState)
Interface	IMeasurement

Read-only **About Marker Search**

FilterBW Property

Description	Returns the results of the SearchBandwidth method.
VB Syntax	*filtBW = meas.***FilterBW**
Variable	**(Type) - Description**
filtBW	**(single)** - Variable to store bandwidth data
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	filterBW = meas.FilterBW 'Read
C++ Syntax	HRESULT get_FilterBW(float* bw)
Interface	IMeasurement

Read-only **About Marker Search**

FilterCF Property

Description	Returns the Center Frequency result of the SearchBandwidth method.
VB Syntax	*filtCF = meas.***FilterCF**
Variable	**(Type) - Description**
filtCF	**(double)** - Variable to store bandwidth CF data
meas	A Measurement **(object)**
Return Type	Double
Default	Not applicable
Examples	filtCF = meas.FilterCF 'Read
C++ Syntax	HRESULT get_FilterCF(double* centerFrequency)
Interface	IMeasurement

Read-only **About Marker Search**

FilterLoss Property

Description	Returns the Loss value of the SearchBandwidth method.
VB Syntax	*filtLoss = meas.***FilterLoss**
Variable	**(Type) - Description**
filtLoss	**(single)** - Variable to store bandwidth Loss data
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	filterLoss = meas.FilterLoss 'Read
C++ Syntax	HRESULT get_FilterLoss(float* loss)
Interface	IMeasurement

Read-only **About Marker Search**

FilterQ Property

Description	Returns the Q (quality factor) result of the SearchBandwidth method.
VB Syntax	*filtQ = meas.***FilterQ**
Variable	**(Type) - Description**
filtQ	**(single)** - Variable to store bandwidth Q data
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	filtQ = meas.FilterQ 'Read
C++ Syntax	HRESULT get_FilterQ(float* quality)
Interface	IMeasurement

Write/Read **About Data Format**

Format Property

Description	Sets or returns the display format of the measurement.
VB Syntax	*meas.***Format** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(enum NADataFormat)** - Choose from:

0 - naDataFormat_**LinMag**
1 - naDataFormat_**LogMag**
2 - naDataFormat_**Phase**
3 - naDataFormat_**Polar**
4 - naDataFormat_**Smith**
5 - naDataFormat_**Delay**
6 - naDataFormat_**Double**
7 - naDataFormat_**Imaginary**
8 - naDataFormat_**SWR**

Return Type Long Integer

Default **1** - naDataFormat_LogMag

Examples
app.TriggerMode = naTriggerModePoint 'Write
fmt = meas.Format 'Read

C++ Syntax
HRESULT get_Format(tagDataFormat *pVal)
HRESULT put_Format(tagDataFormat newVal)

Interface IMeasurement

Write/Read **About Interpolation**

Interpolate Correction Property

Description Turns ON and OFF correction interpolation which calculates new error terms when stimulus values change after calibration.

When this property is ON and error correction is being applied, the calibration subsystem attempts to interpolate the error terms whenever the stimulus parameters are changed.

When this property is OFF under the same circumstances, error correction is turned OFF.

VB Syntax *meas*.**InterpolateCorrection** = *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(boolean)** - Choose from: **True** - Turns correction interpolation ON **False** - Turns correction interpolation OFF

Return Type Boolean

Default True

Examples
meas.InterpolateCorrection = False
callInterpolate = InterpolateCorrection 'Read

C++ Syntax
HRESULT get_InterpolateCorrection(boolean *pVal)
HRESULT put_InterpolateCorrection(boolean newVal)

Interface IMeasurement

Write/Read **About Receiver Cal**

InterpolateNormalization Property

Description	Turns ON and OFF normalization interpolation which calculates new divisor data when stimulus values change after normalization. When this property is ON and normalization is being applied, the Normalization algorithm attempts to interpolate the divisor data whenever the stimulus parameters are changed. When this property is OFF under the same circumstances, normalization is turned OFF. Normalization is currently supported only on measurements of unratioed power for the purpose of performing a receiver power calibration.
VB Syntax	*meas*.**InterpolateNormalization** = *value*
Variable	**(Type) - Description**
meas	**(object) -** A Measurement object
value	**(boolean)** 0 – Turns normalization interpolation OFF 1 – Turns normalization interpolation ON
Return Type	Boolean
Default	0 -OFF
Examples	meas.InterpolateNormalization = 1 'Write normalized = meas.InterpolateNormalization 'Read
C++ Syntax	HRESULT put_InterpolateNormalization(VARIANT_BOOL bState); HRESULT get_InterpolateNormalization(VARIANT_BOOL *bState);
Interface	IMeasurement

Read-only

IsSParameter Property

Description	Returns true if measurement represents an S-Parameter
VB Syntax	*value = meas*.**IsSparameter**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Boolean)** 1 True - measurement is an S-Parameter 0 False - measurement is NOT an S-Parameter
Return Type	Boolean
Default	True
Examples	print app.IsSparameter
C++ Syntax	HRESULT IsSparameter([out, retval] VARIANT_BOOL * bVal);
Interface	IMeasurement2

Read-only **About Limit Testing**

LimitTestFailed Property

Description	Returns the results of limit testing for the measurement.
VB Syntax	*testFailed = meas*.**LimitTestFailed**
Variable	**(Type) - Description**
testFailed	**(boolean)** Variable to store the returned value **False (0**) - Limit Test Passed **True (1**) - Limit Test Failed
meas	A Measurement **(object)**
Return Type	Boolean
Default	False returned if there is no testing in progress
Examples	Dim testRes As Boolean testRes = meas.LimitTestFailed MsgBox (testRes)
C++ Syntax	HRESULT get_LimitTestFailed(VARIANT_BOOL* trueIfFailed)
Interface	IMeasurement

Read-only **About Limit Testing**

LoadPort Property

Description	Returns the load port number associated with an S-parameter reflection measurement. If the measurement is not a reflection S-parameter, the number returned by this property will have no meaning.
VB Syntax	*loadPort = meas.***LoadPort**
Variable	**(Type) - Description**
loadPort	**(long integer) -** The reflection measurement's load port number.
meas	A Measurement **(object)**
Return Type	Long Integer
Default	Not Applicable
Examples	Set meas = pna.ActiveMeasurement loadPort = meas.LoadPort
C++ Syntax	HRESULT get_LoadPort(long *pPortNumber);
Interface	IMeasurement

Write/Read **About Receiver Cal**

LogMagnitudeOffset Property

Description	Sets or returns the power offset value in dBm that the normalized unratioed power measurement data will be shifted by. The unratioed power measurement is effectively calibrated to the power level specified by the value of LogMagnitudeOffset as soon as the Normalization property is set to ON after the DataToDivisor method has been called.
VB Syntax	*meas.***LogMagnitudeOffset** = *value*
Variable	**(Type) - Description**
meas	**(object) -** A Measurement object
value	**(double) -** Power offset in dBm. No limits are enforced on this value, but the PNA receivers themselves have maximum and minimum power specifications. This value must comply with those limits for a valid receiver power calibration
Return Type	Double
Default	0
Examples	meas.LogMagOffset = -10 'Write (-10 dBm) calpower = meas.LogMagOffset 'Read meas.DataToDivisor 'Store meas data as measurement divisor meas.Normalize = 1 'Measurement is now calibrated to –10 dBm
C++ Syntax	HRESULT put_LogMagOffset(double newVal); HRESULT get_LogMagOffset(double *pVal);
Interface	IMeasurement

Write/Read **About Marker Format**

MarkerFormat Property

Description	Sets (or returns) the format of all the markers in the measurement. To override this setting for an individual marker, use **mark.Format**
VB Syntax	*meas.***MarkerFormat** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(enum NAMarkerFormat)** - Choose from: **0** - naMarkerFormat_**LinMag** **1** - naMarkerFormat_**LogMag** **2** - naMarkerFormat_**Phase** **3** - naMarkerFormat_**Delay** **4** - naMarkerFormat_**Real** **5** - naMarkerFormat_**Imaginary** **6** - naMarkerFormat_**SWR** **7** - naMarkerFormat_**LinMagPhase** **8** - naMarkerFormat_**LogMagPhase** **9** - naMarkerFormat_**RealImaginary**

10 - naMarkerFormat_**ComplexImpedance**
11 - naMarkerFormat_**ComplexAdmittance**

Return Type	Long Integer
Default	**1** - naMarkerFormat_LogMag
Examples	meas.MarkerFormat = naMarkerFormat_SWR 'Write fmt = mark.Format 'Read
C++ Syntax	HRESULT put_MarkerFormat(tagNAMarkerFormat NewFormat)
Interface	IMeasurement

Read-only **About Trace Statistics**

Mean Property

Description	Returns the mean value of the measurement . To retrieve all 3 statistics value at the same time, use meas.GetTraceStatistics
VB Syntax	*average = meas.***Mean**
Variable	**(Type) - Description**
average	**(single)** - Variable to store mean value
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	Dim average as Single average = meas.Mean 'Read
C++ Syntax	HRESULT get_Mean(float* mean)
Interface	IMeasurement

Write/Read **About Receiver Cal**

Normalization Property

Description	Sets or returns normalization ON or OFF for the measurement. Normalization is currently supported only on measurements of unratioed power for the purpose of performing a receiver power calibration. If this property is set to ON for a ratioed measurement (such as S-parameter), it will return an error. This property will also return an error when set to ON if the divisor buffer doesn't yet exist.
VB Syntax	*meas.***Normalization** = *value*
Variable	**(Type) - Description**
meas	**(object) -** A Measurement object
value	**(boolean)** 0 – Turns normalization OFF 1 – Turns normalization ON
Return Type	Boolean

Default	0 -OFF
Examples	meas.Normalization = 1 'Write normalized = meas.Normalization 'Read
C++ Syntax	HRESULT put_Normalization(VARIANT_BOOL bState); HRESULT get_Normalization(VARIANT_BOOL *bState);
Interface	IMeasurement

Write/Read **About Traces**

Name (Measurement) Property

Description	Sets (or returns) the Name of the measurement. Measurement names must be unique among the set of measurements. Measurement names cannot be an empty string. **Note**: This is the same name as trace.Name; when one changes, the other changes.
VB Syntax	*meas*.**Name** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(string)** - A user defined name of the measurement
Return Type	String
Default	"CH1_S11_1" - name of the default measurement
Examples	meas.Name = "Filter BPass" 'Write MName = meas.Name 'Read
C++ Syntax	HRESULT get_Name(BSTR *pVal) HRESULT put_Name(BSTR newVal)
Interface	IMeasurement

Read-only **About Measurements**

Number (Measurement) Property

Description	Returns the Number of the measurement. Measurement numbers are assigned internally. **Note**: Measurement numbers are NOT the same as their number in the Measurements collection. Measurement number is used to identify the measurement associated with an event. This property is used to identify measurements when events occur through the OnMeasurementEvent callback. For example: OnMeasurementEvent (naEventId_MSG_LIMIT_FAILED, 3)
VB Syntax	*measNum* = *meas*.**Number**
Variable	**(Type) - Description**

measNum	**(long)** - variable to store the measurement number
meas	A Measurement **(object)**
Return Type	Long Integer
Default	"1" - number of the default measurement
Examples	measNum = meas.Number
C++ Syntax	HRESULT get_Number(long *MeasurementNumber)
Interface	IMeasurement

Read-only

NumberOfPoints Property

Description	Returns the number of data points of the measurement. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value* = *meas*.**NumberOfPoints**
Variable	**(Type) - Description**
value	(**Long**) - variable to store the returned value
meas	A Measurement **(object)**
Return Type	Long Integer
Default	Not Applicable
Examples	Print meas.NumberOfPoints 'prints the number of data points
C++ Syntax	HRESULT get_NumberOfPoints(long *pVal);
Interface	IMeasurement2

Read-only

Parameter Property

Description	Returns the measurement Parameter. To change the parameter, use meas.ChangeParameter
VB Syntax	*measPar* = *meas*.**Parameter**
Variable	**(Type) - Description**
measPar	**(string)** - Variable to store Parameter string
meas	A Measurement **(object)**
Return Type	String
Default	Not applicable
Examples	measPar = meas.Parameter 'Read
C++ Syntax	HRESULT get_Parameter(BSTR *pVal)
Interface	IMeasurement

Read-only **About Trace Statistics**

PeakToPeak Property

Description	Returns the Peak to Peak value of the measurement.To retreive all 3 statistics value at the same time, use meas.GetTraceStatistics
VB Syntax	*pp = meas.***PeakToPeak**
Variable	**(Type) - Description**
pp	**(single)** - Variable to store peak-to-peak value
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	pp = meas.PeakToPeak 'Read
C++ Syntax	HRESULT get_PeakToPeak(float* pp)
Interface	IMeasurement

Write/Read **About Phase Offset**

PhaseOffset Property

Description	Sets the Phase Offset for the active channel.
VB Syntax	*meas.***PhaseOffset** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - PhaseOffset in degrees. Choose any number between: **-360** and **+360**
Return Type	Double
Default	0
Examples	meas.PhaseOffset = 25 'Write poffset = meas.PhaseOffset 'Read
C++ Syntax	HRESULT get_PhaseOffset(double *pVal) HRESULT put_PhaseOffset(double newVal)
Interface	IMeasurement

Read-only

ReceivePort Property

Description	Returns the receiver (response) port number of measurement. To understand how this property is useful, see IMeasurement2 Interface.

VB Syntax	*value* = *meas*.**ReceivePort**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Variable to store the returned value
Return Type	Long Integer
Default	Not Applicable
Examples	rp = meas.ReceivePort
C++ Syntax	HRESULT ReceivePort([out, retval] Long* rcvPort);
Interface	IMeasurement2

Write/Read **About Reference Markers**

ReferenceMarkerState Property

Description	Turn ON or OFF the reference marker. (can you access marker10?)
VB Syntax	*meas*.**ReferenceMarkerState** = *state*
Variable	**(Type) - Description**
app	A Measurement **(object)**
state	(boolean) - ON (1) turns the reference marker ON OFF (0) turns the reference marker OFF
Return Type	Boolean
Default	0 - OFF
Examples	meas.ReferenceMarkerState = True reference = meas.ReferenceMarkerState
C++ Syntax	HRESULT get_ReferenceMarkerState(VARIANT_BOOL bState) HRESULT put_ReferenceMarkerState(VARIANT_BOOL* bState)
Interface	IMeasurement

Write/Read **About Trace Statistics**

ShowStatistics Property

Description	Displays and hides the measurement (Trace) statistics (peak-to-peak, mean, standard deviation) on the screen. To display measurement statistics for a narrower band of the X-axis, use StatisticsRange. The analyzer will display either measurement statistics or Filter Bandwidth statistics; not both.
VB Syntax	*meas*.**ShowStatistics** = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(boolean)** - Boolean value: **1** - Show statistics

0 - Hide statistics

Return Type Boolean

Default 0 - Hide

Examples

meas.ShowStatistics = True 'Write
showstats = meas.ShowStatistics 'Read

C++ Syntax HRESULT put_ShowStatistics(VARIANT_BOOL bState)

Interface IMeasurement

Write/Read **About Smoothing**

SmoothingAperture Property

Description Specifies or returns the amount of smoothing as a ratio of the number of data points in the measurement trace.

VB Syntax *meas*.**SmoothingAperture** = *value*

Variable **(Type) - Description**

meas A Measurement **(object)**

value **(double)** - Smoothing Aperture. A ratio of (aperture points / trace points)/100 Choose any number between **.01** and **.25**.

Return Type Double

Default .25

Examples

meas.**SmoothingAperture** = .10 'Write
saperture = meas.**SmoothingAperture** 'Read

C++ Syntax

HRESULT get_SmoothingAperture(double *pVal)
HRESULT put_SmoothingAperture(double newVal)

Interface IMeasurement

Write/Read **About Smoothing**

Smoothing Property

Description Turns ON and OFF data smoothing.

VB Syntax *meas*.**Smoothing** = *state*

Variable **(Type) - Description**

meas A Measurement **(object)**

state **(boolean)**

1 - Turns smoothing ON
0 - Turns smoothing OFF

Return Type Boolean

Default 0

Examples

meas.Smoothing = 1 'Write

	smooth = meas.Smoothing 'Read
C++ Syntax	HRESULT get_Smoothing(VARIANT_BOOL *pVal) HRESULT put_Smoothing(VARIANT_BOOL newVal)
Interface	IMeasurement

Read-only

SourcePort Property

Description	Returns the source port of measurement. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value = meas*.**SourcePort**
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Variable to store the returned value
Return Type	Long Integer
Default	Not Applicable
Examples	sp = meas.SourcePort
C++ Syntax	HRESULT SourcePort([out, retval] Long* srcPort);
Interface	IMeasurement2

Read-only

Span Property

Description	Returns the stimulus span of the measurement (stop-start data points). To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value = meas*.**Span**
Variable	**(Type) - Description**
value	**(Double)** - Variable to store the returned value.
meas	A Measurement **(object)**
Return Type	Double
Default	Not Applicable
Examples	Print meas.Span 'prints the span of the measurement
C++ Syntax	HRESULT get_Span(double * Val);
Interface	IMeasurement2

Read-only **About Trace Statistics**

StandardDeviation Property

Description	Returns the standard deviation of the measurement. To retreive all 3 statistics value at the same time, use meas.GetTraceStatistics
VB Syntax	*stdev* = *meas*.**StandardDeviation**
Variable	**(Type) - Description**
stdev	**(single)** - Variable to store standard deviation value
meas	A Measurement **(object)**
Return Type	Single
Default	Not applicable
Examples	stdev = meas.StandardDeviation 'Read
C++ Syntax	HRESULT get_StandardDeviation(float* stdDeviation)
Interface	IMeasurement

Read-only

Start Property

Description	Returns the stimulus value of the first data point for the measurement. To understand how this property is useful, see IMeasurement2 Interface.
VB Syntax	*value* = *meas*.**Start**
Variable	**(Type) - Description**
value	**(Double)** - Variable to store the returned value
meas	A Measurement **(object)**
Return Type	Double
Default	Not Applicable
Examples	Print meas.Start 'prints the stimulus value of the first data point
C++ Syntax	HRESULT get_Start (double * Val);
Interface	IMeasurement2

Write/Read **About User Ranges**

Statistics Range Property

Description	Sets the User Range number for calculating measurement statistics. Set the start and stop values for a User Range with chan.UserRangeMin and chan.UserRangeMax. There are 9 User Ranges per channel. User ranges are applied independently to any measurement.
VB Syntax	*meas*.**StatisticsRange** = *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(long integer)** - Range Number. Choose any number between 0 and 9. **1 - 9** are user-defined ranges **0** is Full Span
Return Type	Long Integer
Default	0

Examples

meas.StatisticsRange = 2 'Write
statrange = meas.StatisticsRange 'Read

C++ Syntax

HRESULT get_StatisticsRange(long* rangeNumber)
HRESULT put_StatisticsRange(long rangeNumber)

Interface IMeasurement

Read- only

Stop Property

Description Returns the stimulus value of the last data point for the measurement. To understand how this property is useful, see IMeasurement2 Interface.

VB Syntax *value* = *meas*.**Stop**

Variable	**(Type) - Description**
value	**(Double)** Variable to store the returned value
meas	A Measurement **(object)**
Return Type	Double
Default	Not Applicable

Examples Print meas.Stop 'prints the stimulus value of the last data point

C++ Syntax HRESULT get_Stop(double * Val);

Interface IMeasurement2

Write/Read **About Math Operations**

TraceMath Property

Description Performs math operations on the measurement object and the trace stored in memory. (There MUST be a trace stored in Memory to perform math. See Meas.DataToMemory method.)

VB Syntax *meas.***TraceMath** = *value*

Variable	**(Type) - Description**
meas	A measurement **(object)**
value	**(enum NAMathOperation)** - Choose from: **0** - naDataNormal **1** - naDataMinusMemory

2 - naDataPlusMemory
3 - naDataDivMemory
4 - naDataTimesMemory

Return Type NAMathOperation
Default Normal (0)

Examples
meas.TraceMath = naDataMinusMemory 'Write
mathOperation = meas.TraceMath 'Read

C++ Syntax
HRESULT get_TraceMath(tagNAMathOperation* pMathOp)
HRESULT put_TraceMath(tagNAMathOperation mathOp)

Interface IMeasurement

Write/Read **About Math Operations**

View Property

Description Sets (or returns) the type of trace displayed on the screen.
VB Syntax *meas.***View** = *value*

Variable	(Type) - Description
meas	A measurement **(object)**
value	**(enum NAView)** - Type of trace. Choose from: **0** - naData **1** - naDataAndMemory **2** - naMemory **3** - naNoTrace **Note:** The **naData** trace may reflect the result of a TraceMath operation.

Return Type NAView
Default naData

Examples
meas.View = naData 'Write
trceview = meas.View 'Read

C++ Syntax
HRESULT get_View(tagNAView* pView)
HRESULT put_View(tagNAView newView)

Interface IMeasurement

Measurement Object custom IMixer Interface

Write/Read

LoadFile Method

Description Loads a previously-configured mixer attributes file (.mxr)
VB Syntax *value = meas.***LoadFile()**

Variable	(Type) - Description
value	**(string)**

meas	A Measurement **(object)**
Return Type	string
Default	Not Applicable
Examples	Print meas.LoadFile("MyMixer.mxr") 'prints the value of the external source.
C++ Syntax	HRESULT get_LoadFile(BSTR *pVal) HRESULT put_LoadFile(BSTR newVal)
Interface	IMixer

Write/Read

SaveFile Method

Description	Saves the settings for the mixer/converter test setup to a mixer attributes file.
VB Syntax	*value* = *meas*.**SaveFile()**
Variable	**(Type) - Description**
value	(string)
meas	A Measurement **(object)**
Return Type	string
Default	Not Applicable
Examples	Print meas.SaveFile("MyMixer.mxr")
C++ Syntax	HRESULT get_SaveFile(BSTR *pVal) HRESULT put_SaveFile(BSTR newVal)
Interface	IMixer

Write-Read

ElecDelayMedium Property

Description	Sets or returns the characteristic of the electrical delay medium.
VB Syntax	*meas*.**ElecDelayMedium =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(enum NACalStandardMedium)** choose from 0 - naCoax 1 - naWaveGuide
Return Type	NACalStandardMedium
Default	Not Applicable
Examples	Print meas.ElecDelayMedium 'prints the value of the electrical delay medium
C++ Syntax	HRESULT get_OutputStopFrequency(NACalStandardMedium *pVal) HRESULT put_OutputStopFrequency(NACalStandardMedium

	newVal)
Interface	IMixer

Write/Read

Fixed Frequency Property

Description	Sets or returns the frequency fixed value
VB Syntax	*meas*.**FixedFrequency =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	Print meas.FixedFrequency 'prints the value of the fixed frequency meas.FixedFrequency = 1e9 'Set to 1 Ghz
C++ Syntax	HRESULT get_FixedFrequency(double *pVal) HRESULT put_FixedFrequency(double newVal)
Interface	IMixer

Write/Read

IFDenominator Property

Description	Sets or returns the denominator value of the IF Fractional Multiplier. **Note:** This command only applies to 2 stage mixers
VB Syntax	*meas*.**IFDenominator =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable
Examples	Print meas.IFDenominator 'prints the value of the IFDenominator
C++ Syntax	HRESULT get_IFDenominator(Long *pVal) HRESULT put_IFDenominator(Long newVal)
Interface	IMixer

Write/Read

IFNumerator Property

Description	Sets or returns the numerator value of the IF Fractional Multiplier. **Note:** This command only applies to 2 stage mixers
VB Syntax	*meas*.**IFNumerator =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable

Examples	Print meas.IFNumerator 'prints the value of the IFNumerator

C++ Syntax	HRESULT get_IFNumerator(Long *pVal) HRESULT put_IFNumerator(Long newVal)
Interface	IMixer

Write/Read

IFSideband Property

Description	Sets or returns the value of the IF sideband, high or low. **Note:** This command only applies to 2 stage mixers
VB Syntax	*meas*.**IFSideband =***value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(FCASideBand)** - High or Low.
Return Type	FCASideBand
Default	Not Applicable

Examples	Print meas.IFSideband 'prints the value of the IFSideband

C++ Syntax	HRESULT get_IFSideband(FCASideBand *pVal) HRESULT put_IFSideband(FCASideBand newVal)
Interface	IMixer

Write/Read

IFStartFrequency Property

Description	Sets or returns the start frequency value of the mixer IF frequency. **Note:** This command only applies to 2 stage mixers
VB Syntax	*meas*.**IFStartFrequency =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.

Return Type Double
Default Not Applicable

Examples Print meas.IFStartFrequency 'prints the value of the IFStartFrequency

C++ Syntax HRESULT get_IFStartFrequency(double *pVal)
HRESULT put_IFStartFrequency(double newVal)
Interface IMixer

Write/Read

IFStopFrequency Property

Description Sets or returns the stop frequency value of the mixer IF frequency.
Note: This command only applies to 2 stage mixers
VB Syntax *meas*.**IFStopFrequency =** *value*

Variable	(Type) - Description
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable

Examples Print meas.IFStopFrequency 'prints the value of the IFStopFrequency

C++ Syntax HRESULT get_IFStopFrequency(double *pVal)
HRESULT put_IFStopFrequency(double newVal)
Interface IMixer

Write/Read **About Port Extensions**

InputC Property

Description Sets the Port Extension value for Receiver C
VB Syntax *portExt*.**InputC** = *value*

Variable	(Type) - Description
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0

Examples
portExt.InputC = 10e-6 'Write
inC = portExt.InputC 'Read

C++ Syntax HRESULT get_InputC(double *pVal)
HRESULT put_InputC(double newVal)
Interface IPortExtension

Write/Read

InputDenominator Property

Description	Sets or returns the denominator value of the Input Fractional Multiplier.
VB Syntax	*meas*.**InputDenominator =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable
Examples	Print meas.InputDenominator 'prints the value of the InputDenominator
C++ Syntax	HRESULT get_InputDenominator(long *pVal) HRESULT put_InputDenominator(long newVal)
Interface	IMixer

Write/Read

InputNumerator Property

Description	Sets or returns the numerator value of the Input Fractional Multiplier.
VB Syntax	*meas*.**InputNumerator =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable
Examples	Print meas.InputNumerator 'prints the value of the InputNumerator
C++ Syntax	HRESULT get_InputNumerator(long *pVal) HRESULT put_InputNumerator(long newVal)
Interface	IMixer

Write/Read

InputPower Property

Description	Sets or returns the value of the Input Power.
VB Syntax	*meas*.**InputPower =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Power in dB.
Return Type	Double
Default	Not Applicable

Examples Print meas.InputPower 'prints the value of the InputPower

C++ Syntax HRESULT get_InputPower(double *pVal)
HRESULT put_InputPower(double newVal)

Interface IMixer

Write/Read

InputStartFrequency Property

Description	Sets and returns the start frequency value of the mixer Input frequency.
VB Syntax	*meas*.**InputStartFrequency =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable

Examples Print meas.InputStartFrequency 'prints the value of the InputStartFrequency

C++ Syntax HRESULT get_InputStartFrequency(double *pVal)
HRESULT put_InputStartFrequency(double newVal)

Interface IMixer

Write/Read

InputStopFrequency Property

Description	Sets and returns the stop frequency value of the mixer Input frequency.
VB Syntax	*meas*.**InputStopFrequency =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable

Examples	Print meas.InputStoptFrequency 'prints the value of the InputStopFrequency
C++ Syntax	HRESULT get_InputStopFrequency(double *pVal) HRESULT put_InputStopFrequency(double newVal)
Interface	IMixer

Write/Read

LODenominator Property

Description	Sets or returns the denominator value of the LO Fractional Multiplier.
VB Syntax	*meas*.**LODenominator** n **=** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
<n>	Long, index of the LO number, 1 or 2
Return Type	Long
Default	Not Applicable
Examples	Print meas.LODenominator 2 'prints the value of the second LODenominator
C++ Syntax	HRESULT get_LODenominator(long *pVal) HRESULT put_LODenominator(long newVal)
Interface	IMixer

Write/Read

LOFixedFrequency Property

Description	Sets or returns the LO frequency fixed value
VB Syntax	*meas*.**LOFixedFrequency** n = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
n	**Long**, index of the LO number, 1 or 2
Return Type	Double
Default	Not Applicable
Examples	Print meas.LOFixedFrequency 2 'prints the value of the second LO fixed frequency

C++ Syntax	HRESULT get_LOFixedFrequency(double *pVal) HRESULT put_LOFixedFrequency(double newVal)
Interface	IMixer

Write/Read

LOName Property

Description	Sets or returns the LO name
VB Syntax	*meas*.**LOName =***value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(string)** - Frequency in Hertz.
Return Type	string
Default	Not Applicable
Examples	Print meas.LOName 'prints the value of the LO name
C++ Syntax	HRESULT get_LOName(string *pVal) HRESULT put_LOName(string newVal)
Interface	MIXer

Write/Read

LONumerator Property

Description	Sets or returns the numerator value of the LO Fractional Multiplier.
VB Syntax	*meas*.**LONumerator** n = *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
n	**Long**, index of the LO number, 1 or 2
Return Type	Long
Default	Not Applicable
Examples	Print meas.LONumerator 2 'prints the value of the second LO Numerator
C++ Syntax	HRESULT get_LONumerator(long *pVal) HRESULT put_LONumerator(long newVal)
Interface	IMixer

Write/Read

LOPower Property

Description	Sets or returns the value of LO Power.
VB Syntax	*meas*.**LOPower =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Power in dB.
Return Type	Double
Default	Not Applicable
Examples	Print meas.LOPower 'prints the value of the LO Power
C++ Syntax	HRESULT get_LOPower(double *pVal) HRESULT put_LOPower(double newVal)
Interface	IMixer

Write/Read

LOStage Property

Description	Sets or returns the stage value of the mixer
VB Syntax	*meas*.**LOStage =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - **1** or **2** stage mixer.
Return Type	Long
Default	Not Applicable
Examples	Print meas.LOStage 'prints the value of the LO stage
C++ Syntax	HRESULT get_LOStage(long *pVal) HRESULT put_LOStage(long newVal)
Interface	IMixer

Write/Read

OutputDenominator Property

Description	Sets or returns the denominator value of the Output Fractional Multiplier.
VB Syntax	*meas*.**OutputDenominator =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable

Examples Print meas.OutputDenominator 'prints the value of the OutputDenominator

C++ Syntax HRESULT get_OutputDenominator(long *pVal)
HRESULT put_OutputDenominator(long newVal)

Interface IMixer

Write/Read

OutputNumerator Property

Description Sets or returns the numerator value of the Output Fractional Multiplier.

VB Syntax *meas*.**OutputNumerator =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(Long)** - Frequency in Hertz.
Return Type	Long
Default	Not Applicable

Examples Print meas.OutputNumerator 'prints the value of the OutputNumerator

C++ Syntax HRESULT get_OutputNumerator(long*pVal)
HRESULT put_OutputNumerator(long newVal)

Interface IMixer

Write/Read

OutputSideband Property

Description Sets or returns the value of the output sideband, high or low.

VB Syntax *meas*.**OutputSideband =** *value*

Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(FCASideBand)** - High or Low.
Return Type	FCASideBand
Default	Not Applicable

Examples Print meas.OutputSideband 'prints the value of the OutputSideband

C++ Syntax	HRESULT get_OutputSideband(FCASideBand *pVal) HRESULT put_OutputSideband(FCASideBand newVal)
Interface	IMixer

Write/Read

OutputStartFrequency Property

Description	Returns the start frequency value of the mixer output start frequency.
VB Syntax	*meas*.**OutputStartFrequency =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	Print meas.OutputStartFrequency 'prints the value of the OutputStartFrequency
C++ Syntax	HRESULT get_OutputStartFrequency(double *pVal) HRESULT put_OutputStartFrequency(double newVal)
Interface	IMixer

Write/Read

OutputStopFrequency Property

Description	Returns the start frequency value of the mixer Output Stop frequency.
VB Syntax	*meas*.**OutputStopFrequency =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	Print meas.OutputStopFrequency 'prints the value of the OutputStopFrequency
C++ Syntax	HRESULT get_OutputStopFrequency(double *pVal) HRESULT put_OutputStopFrequency(double newVal)
Interface	IMixer

Write-Read

WGCutoffFreq Property

Description	Sets or returns the value of the waveguide cut off frequency.
VB Syntax	*meas*.**WGCutoffFreq =** *value*
Variable	**(Type) - Description**
meas	A Measurement **(object)**
value	**(double)** - Frequency in Hertz.
Return Type	Double
Default	Not Applicable
Examples	Print meas.WGCutoffFreq 'prints the value of the waveguide cut off frequency
C++ Syntax	HRESULT get_OutputStopFrequency(double *pVal) HRESULT put_OutputStopFrequency(double newVal)
Interface	IMixer

Measurent Object custom IArrayTransfer Interface

Read-only | Data Access Map

GetComplex Method

Description	Retrieves complex data from the specified location. See also getNAComplex , getData , and getPairedData Methods
VB Syntax	*measData*.**getComplex** *location, numPts, real(), imag()*
Variable	**(Type) - Description**
measData	An IArrayTransfer interface which supports the Measurement object
location	**(enum NADataStore - IArrayTransfer)** - Where the data you want is residing. Choose from: 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
numPts	**(long integer)** - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the arrays
real	**(single)** - Array to store the real values
imag	**(single)** - Array to store the imaginary values
Return Type	Single
Default	Not Applicable
Examples	Dim real(201) AS Single Dim imag(201) AS Single

```
 Dim pts as Integer
 Dim measData As IArrayTransfer
 Set measData = app.ActiveMeasurement
 measData.getComplex naCorrectedData, pts, real(0), imag(0)
```

C++ Syntax IArrayTransfer - HRESULT getComplex(tagNADataStore DataStore, long* pNumValues, float* pReal, float* pImag)

Interface IArrayTransfer

Read-only **Data Access Map**

GetNAComplex Method

Description Retrieves complex data from the specified location. See also getComplex and getData Method.

VB Syntax *measData*.**getNAComplex** *location, numPts, data*

Variable	**(Type) - Description**
measData	An IArrayTransfer interface which supports the Measurement object
location	**(enum NADataStore)** - Where the data you want is residing. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
numPts	**(long integer)** - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the *dComplex* array
data	**(NAComplex)** - A one-dimensional array of NaComplex to store the data.
Return Type	NAComplex
Default	Not Applicable

Examples

```
Dim dComplex(201) AS NaComplex
 Dim measData As IArrayTransfer
 Dim pts as Long
 Set measData = app.ActiveMeasurement
 measData.getNAComplex naCorrectedData, pts, dComplex(0)
```

Notes The data is stored as Real and Imaginary (**Re** and **Im**) members of the NaComplex user defined type. You can access each number individually by iterating through the array.

```
For i = 0 to NumPts-1
  dReal (i) = dcomplex (i).Re
  dImag (i) = dcomplex (i).Im
 Next i
```

C++ Syntax HRESULT getNAComplex(tagNADataStore DataStore, long* pNumValues, TsComplex* pComplex)

Interface IArrayTransfer

Read-only **Data Access Map**

GetPairedData Method

Description	Retrieves pairs of data from the specified location. **Note:** This method exists on a non-default interface. If you cannot access this method, use the Get Data Method on IMeasurement.
VB Syntax	*measData*.**getPairedData** *location, format, numPts, d1, d2*
Variable	**(Type) - Description**
measData	An IArrayTransfer interface which supports the Measurement object
location	**(enum NADataStore)** - Where the data you want is residing. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
format	**(enum NAPairedDataFormat)** - Format in which you would like the Paired data. Choose from: • **naLinMagPhase** - Linear magnitude and phase • **naLogMagPhase** - Log magnitude and phase • **naRealImaginary** - Real and Imaginary **Note**: Selecting **naRealImaginary** format is the same as using the getComplex method
numPts	**(long integer)** - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the *dPaired* array
d1	**(single)** - Array to store the magnitude / real values
d2	**(single)** - Array to store the phase / imaginary values
Return Type	Two Single arrays
Default	Not Applicable

Examples

```
Dim logm() As Single
 Dim phase() As Single
 Public measData As IArrayTransfer
 Set measData = app.ActiveMeasurement
 Dim numpts As Long
 numPoints = app.ActiveChannel.NumberOfPoints
 ReDim logm(numPoints)
 ReDim phase(numPoints)

 measData.getPairedData naCorrectedData, naLogMagPhase,
numPoints, logm(0), phase(0)

 Print values(0), values(1)
```

C++ Syntax

HRESULT getPairedData(tagNADataStore DataStore, tagNAPairedDataFormat PairFormat, long* pNumValues, float* pReal, float* pImag)

Interface IArrayTransfer

Read-only **Data Access Map**

GetScalar Method

Description

Retrieves scalar data from the specified locations.

Note: This method exists on a non-default interface. If you cannot access this method, use the Get Data Method on IMeasurement.

Note: You can **NOT** use this command to get complex data.

VB Syntax *measData*.**getScalar** *location, format, numPts, data*

Variable **(Type) - Description**

measData An IArrayTransfer interface which supports the Measurement object

location **(enum NADataStore)** - Where the data you want is residing. Choose from:

0 - naRawData
1 - naCorrectedData
2 - naMeasResult
3 - naRawMemory
4 - naMemoryResult
5 - naDivisor

See the Data Access Map

format **(enum DataFormat)** - Scalar format in which you would like the data. Choose from:

- naDataFormat_Delay
- naDataFormat_Imaginary
- naDataFormat_LinMag
- naDataFormat_LogMag
- naDataFormat_Phase
- naDataFormat_Real

naDataFormat_SWR

numPts **(long integer)** - Number of data points requested
[out] - specifies number of data elements returned
[in] - specifies the data being requested or the capacity of the *dScalar* array

data **(single)** - Array to store the scalar data.

Return Type Single

Default Not Applicable

Examples

```
Dim dScalar() As Single
Dim measData As IArrayTransfer
Set measData = app.ActiveMeasurement
Dim numpts as Long
numpts = app.ActiveChannel.NumberOfPoints
ReDim dScalar(numPoints)

measData.getScalar naCorrectedData, naDataFormat_LogMag, numpts,
```

dScalar(0)
 Print dScalar(0), dScalar(1)

C++ Syntax HRESULT getScalar(tagNADataStore DataStore, tagDataFormat DataFormat, long* pNumValues, float* pVals)

Interface IArrayTransfer

Write-only **Data Access Map**

PutComplex Method

Description

Puts real and imaginary data into the specified location. This method forces the channel into Hold mode to prevent the input data from being overwritten. Learn more about reading and writing Cal Data using COM.

Data put in the raw data store will be **re-processed** whenever a change is made to the measurement attributes such as format or correction.

Data put in the measurement results store will be **overwritten** by any measurement attribute changes.

See also putNAComplex

VB Syntax *measData*.**putComplex** *location, numPts, real(), imag(),* [*format*]

Variable	(Type) - Description
measData	An IArrayTransfer interface which supports the Measurement object
location	**(enum NADataStore)** Where the Data will be put. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
numPts	**(long integer)** - Number of data points in the channel
real()	**(single) -** Array containing real data values
imag()	**(single)** -Array containing imaginary data values
format	**(enum NADataFormat)** optional argument - display format of the real and imaginary data. Only used if destination is naMeasResult or naMemoryResult buffer. If unspecified, data is assumed to be in naDataFormat_Polar • naDataFormat_Delay • naDataFormat_Imaginary • naDataFormat_LinMag • naDataFormat_LogMag • naDataFormat_Phase • naDataFormat_Real • naDataFormat_SWR • naDataFormat_Smith naDataFormat_Polar

Return Type Not Applicable

Default Not Applicable

Examples

```
Dim measData As IArrayTransfer
 Set measData = app.ActiveMeasurement

 measData.putComplex naMemoryResult, 201,
real(0),imag(0),naDataFormat_SWR
```

C++ Syntax HRESULT putComplex(tagNADataStore DataStore, long lNumValues, float* pReal, float* pImag, tagDataFormat displayFormat)

Interface IArrayTransfer

Write-only **Data Accessing Map**

PutNAComplex Method

Description

Puts complex data into the specified location. This method forces the channel into Hold mode to prevent the input data from being overwritten. The data is processed and displayed.

Data put in the naRawData store will be **re-processed** whenever a change is made to the measurement attributes such as format or correction.

Data put in the naMeasResult store will be **overwritten** by any measurement attribute changes (such as moving a marker).

Note: This method uses NAComplex which is a user-defined data type. If you cannot or prefer not to use this data type, use the putComplex method.

VB Syntax *measData*.**putNAComplex** *location, numPts, data,* [*format*]

Variable	**(Type) - Description**
measData	An IArrayTransfer interface which supports the Measurement object
location	**(enum NADataStore)** Where the Data will be put. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor See the Data Access Map
numPts	**(long integer)** - Number of data points in the channel
data	**(NAComplex)** - A one-dimensional array of Complex data matching the number of points in the current measurement.
format	**(enum NADisplayFormat) -** Optional argument. Format of the data. If unspecified, naDataFormat_Polar is assumed. Only used when the destination store is naMeasResult or naMemoryResult.

Return Type Not Applicable

Default Not Applicable

Examples

```
Dim measData As IArrayTransfer
 Set measData = app.ActiveMeasurement

 measData.putNAComplex naMemoryResult, 201, dRawComplex(0)
```

C++ Syntax HRESULT putNAComplex(tagNADataStore DataStore, long lNumValues, TsComplex* pArrayOfComplex, tagDataFormat displayFormat)

Interface IArrayTransfer

Write-only **Data Access Map**

PutScalar Method

Description Puts Scalar data in the Measurement Result buffer. The putScalar array is not processed by the analyzer; it is just displayed. Any change to the measurement state (changing the format, for example) will cause the putScalar data to be overwritten with the data processed from the raw data buffer.

VB Syntax *measData*.**putScalar***, format, numPts, data*

Variable	**(Type) - Description**
measData	An IArrayTransfer interface which supports the Measurement object.
format	**(enum NADataFormat)** Format of the data. Choose from: 1 - naDataFormat_LinMag 2 - naDataFormat_LogMag 3 - naDataFormat_Phase 6 - naDataFormat_Delay 7 - naDataFormat_Real 8 - naDataFormat_Imaginary 9 - naDataFormat_SWR **Note: Smith and Polar** formats are not allowed. See the Data Access Map
numPts	**(integer)** - Number of values. Usually the number of points in the trace (chan.NumberOfPoints).
data	**(single)** - A one-dimensional array of Scalar data matching the number of points in the current measurement.

Return Type Not Applicable

Default Not Applicable

Examples

```
Dim measData As IArrayTransfer
 Set measData = app.ActiveMeasurement

 measData.putScalar naDataFormat_LogMag, 201, dScalar(0)
```

C++ Syntax HRESULT putScalar\(tagDataFormat eFormat, long lNumValues, float* pArrayOfScalar)

Interface IArrayTransfer

NAWindows Collection

NAWindows Collection

Description

A collection object that provides a mechanism for iterating through the Application windows. See Collections in the Analyzer.

Methods	Description
Add	Adds a window to the NAWindows collection.
Item	Use to get a handle to a channel in the collection.
Remove	Removes a window from the NAWindows collection.
Properties	**Description**
Count	Returns the number of windows on the analyzer.
Parent	Returns a handle to the current Application.

Write-only **About Windows**

Add (NAWindows) Method

Description	Add a window to the display. Does not add a measurement. The window number must not already exist.
VB Syntax	*wins.***Add** *[item]*
Variable	**(Type) - Description**
wins	A NAWindow collection **(object)**
item	**(variant)** - optional argument; Window number. Range between 1 - 4
Return Type	Object
Default	Not Applicable
Examples	wins.Add 3 'Creates a window number 3
C++ Syntax	HRESULT Add(long windowNumber)
Interface	INAWindows

NAWindow Object

NAWindow Object (default interface is INAWindow)

Description

The NAWindow object controls the part of the display that contains the graticule, or what is written on the display.

Methods	Description
Autoscale	Autoscales all measurements in the window. Shared with the Trace Object
ShowMarkerReadout	Shows and Hides the Marker readout for the active marker in the upper-right corner of the window object.
ShowTable	Shows or Hides the specified table for the active measurement in the lower part of the window object.
Property	**Description**
ActiveTrace	Sets a trace to the Active Trace.
MarkerReadout	Sets and reads the state of the Marker readout for the active marker

	in the upper-right corner of the window object.
MarkerReadoutSize	Specifies the size of font used when displaying Marker readout in the selected window.
OneMarkerReadoutPerTrac e	Either show marker readout of only the active trace or all of the traces simultaneously.
Title	Writes or reads a custom title for the window.
TitleState	Turns ON and OFF the window title.
Traces (collection)	
WindowNumber	Reads the number of the active window.
WindowState	Maximizes or minimizes a window.
	Shared with the Application Object

Write-only **About Display Formatting**

Autoscale Method

Description	Autoscales the trace (Trace object) or all of the traces (NAWindow object).
VB Syntax	*object.***Autoscale**
Variable	**(Type) - Description**
object	Trace **(object)** **or** NAWindow **(object)**
Return Type	Not Applicable
Default	Not Applicable
Examples	Trac.Autoscale 'Autoscales the trace Win.Autoscale 'Autoscales all the traces in the window -Write
C++ Syntax	HRESULT AutoScale()
Interface	INAWindow ITrace

Write-only **About Display Formatting**

ShowMarkerReadout Method

Description	Shows and Hides the Marker readout for the active marker in the upper-right corner of the window.
VB Syntax	*win.***ShowMarkerReadout** *state*
Variable	**(Type) - Description**
win	A NAWindow **(object)**
state	**(boolean)** - **True (1)** - Show the Marker readout **False (0)** - Hide the Marker readout
Return Type	Not Applicable
Default	Not Applicable

Examples	win.ShowMarkerReadout True
C++ Syntax **Interface**	HRESULT ShowMarkerReadout(VARIANT_BOOL bState) INAWindow

Write-only **About Display Formatting**

ShowTable Method

Description	Shows or Hides the specified table for the window's active measurement in the lower part of the window.
VB Syntax	*win.***ShowTable** *value*
Variable	**(Type) - Description**
win	A NAWindow **(object)**
value	**(enum naTable)** - The table to show or hide. Choose from: 0 - naTable_None 1 - naTable_Marker 2 - naTable_Segment 3 - naTable_Limit
Return Type	Not Applicable
Default	Not Applicable
Examples	win.ShowTable naTable_limit
C++ Syntax **Interface**	HRESULT ShowTable (tagNATableType table) INAWindow

Read-only **About Traces**

ActiveTrace Property

Description	Returns a handle to the Active Trace object. You can either **(1)** use the handle directly to access trace properties and methods, or **(2)** set a variable to the trace object. The variable retains a handle to the original trace if another trace becomes active.
VB Syntax	1) *win.***ActiveTrace***.<setting>* **or** *2)* Set *trce = win.***ActiveTrace**
Variable	**(Type) - Description**
trce	A Trace **(object)**
win	An NAWindow **(object)**
<setting>	A trace property (or method) and arguments
Return Type	An NAWindow object
Default	None
Examples	1) win.ActiveTrace.Autoscale 2) Public trce as Trace Set trce = Application.ActiveNAWindow.ActiveTrace
C++ Syntax	HRESULT get_ActiveTrace(ITrace* *pVal)

Interface	INAWindow

Write/Read **About Marker Readout**

MarkerReadout Property

Description	Enables or disables the readout of markers in the window. To show the marker on the screen use ShowMarkerReadout Method.
VB Syntax	*win*.**MarkerReadout** = *state*
Variable	**(Type) - Description**
win	A NAWindow (object)
state	**(boolean)** **True** (1) - enables marker readout **False** (0) - disables marker readout
Return Type	Boolean
Default	True
Examples	`win.MarkerReadout = True 'Write` State = app.ActiveNAWindow.MarkerReadout 'Read
C++ Syntax	HRESULT get_MarkerReadout(VARIANT_BOOL *pVal) HRESULT put_MarkerReadout(VARIANT_BOOL newVal)
Interface	INAWindow

Write/Read **About Marker Readout**

MarkerReadoutSize Property

Description	Specifies the size of font used when displaying Marker Readout in the selected window.
VB Syntax	*win*.**MarkerReadoutSize** = *value*
Variable	**(Type) - Description**
win	A NAWindow (object)
value	**(enum NAFontSize)** **0 - naDefault** - marker readout appears in default font size **1 - naLarge** - marker readout appears in large font size
Return Type	Long Integer
Default	naDefault
Examples	`win.MarkerReadoutSize = naDefault 'write default size for marker readout` Dim Size As NAFontSize Size = app.ActiveNAWindow.MarkerReadoutSize 'Read

C++ Syntax HRESULT get_MarkerReadoutSize(tagNAFontSize *pVal)
HRESULT put_MarkerReadoutSize(tagNAFontSize newVal)

Interface INAWindow

Write/Read **About Marker Readout**

OneReadoutPerTrace Property

Description Either show marker readout of only the active trace or all of the traces simultaneously.

VB Syntax *win*.**OneReadoutPerTrace** = *state*

Variable	(Type) - Description
win	A NAWindow (object)
value	**(boolean)** **True** (1) - show a single marker per trace **False** (0) - show up to 4 markers per active trace
Return Type	Boolean
Default	False (0)

Examples

```
win.OneReadoutPerTrace = True 'Write
State = app.ActiveNAWindow.OneReadoutPerTraceBegResp 'Read
```

C++ Syntax HRESULT get_OneReadoutPerTrace(VARIANT_BOOL *pVal)
HRESULT put_OneReadoutPerTrace(VARIANT_BOOL newVal)

Interface INAWindow

Write/Read **About Title**

Title Property

Description Writes or reads a custom title for the window. Newer entries replace (not append) older entries.Turn the title ON and OFF with TitleState

VB Syntax *win.***Title** = *string*

Variable	(Type) - Description
win	A NaWindow **(object)**
string	**(long)** - Title limited to 50 characters.
Return Type	String
Default	Null

Examples

```
win.Title = "Hello World" 'Write
titl = win.Title 'Read
```

C++ Syntax HRESULT get_Title(BSTR *title)
HRESULT put_Title(BSTR title)

Interface INAWindow

Write/Read **About Titles**

TitleState Property

Description	Turns ON and OFF the window title. Write a window title with Title
VB Syntax	*win.***TitleState** = *state*
Variable	**(Type) - Description**
win	A NaWindow **(object)**
state	**(boolean)** **True (1)** - Title ON **False (0)** - Title OFF
Return Type	Long Integer **0** - Title OFF **1** - Title ON
Default	0 - OFF
Examples	win.TitleState = True 'Write titlestate = win.TitleState 'Read
C++ Syntax	HRESULT get_TitleState(VARIANT_BOOL* bState) HRESULT put_TitleState(VARIANT_BOOL bState)
Interface	INAWindow

Read-only

WindowNumber Property

Description	Returns the window number. You might use this property to identify a particular window so that you can create a new Measurement in that window.
VB Syntax	*value* = *win.***WindowNumber**
Variable	**(Type) - Description**
win	A NAWindow (object)
value	**(long integer) -** Variable to store the returned window number
Return Type	Long Integer
Default	Not Applicable
Examples	value = app.ActiveNAWindow.WindowNumber
C++ Syntax	HRESULT (long* windowNumber);
Interface	INAWindow

Write/Read **About Arranging Windows**

WindowState Property

Description	Sets or returns the window setting of Maximized, Minimized, or Normal. To arrange all of the windows, use app.ArrangeWindows.
VB Syntax	*object*.**WindowState** = *value*
Variable	**(Type) - Description**
object	An Application **(object)** - main window **or** A NaWindow **(object)** - data windows
value	**(enum NAWindowStates)** - The window state. Choose from: **0 - naMinimized** - Minimizes the window to an Icon on the lower toolbar **1 - naMaximized** - Maximizes the window **2 - naNormal** - changes the window size to the user defined setting (between Max and Min).
Return Type	Long Integer
Default	naMaximized
Examples	app.WindowState = naMinimized 'changes the Network Analyzer application window to an icon. -Write win.WindowState = naNormal 'changes the window defined by the win object variable to user defined settings. -Write winstate = app.WindowState 'Read
C++ Syntax	HRESULT get_WindowState(tagNAWindowStates *pVal) HRESULT put_WindowState(tagNAWindowStates newVal)
Interface	INAWindow IApplication

Port Extension Object

Port Extensions Object (default interface is IPortExtension)

Description

Contains the methods and properties that control Port Extensions.

Methods

None

Property	**Description**
Input A	Sets the Input A extension value.
Input B	Sets the Input B extension value.
Input C	Sets the Input C extension value.
Port 1	Sets the Port 1 extension value.
Port 2	Sets the Port 2 extension value.
Port 3	Sets the Port 3 extension value.
State	Turns Port Extensions ON and OFF.

Write/Read **About Port Extensions**

InputA Property

Description	Sets a Port Extension value for Receiver A
VB Syntax	*portExt*.**InputA** = *value*
Variable	**(Type) - Description**
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0
Examples	portExt.InputA = 10e-6 'Write inA = portExt.InputA 'Read
C++ Syntax	HRESULT get_InputA(double *pVal) HRESULT put_InputA(double newVal)
Interface	IPortExtension

Write/Read **About Port Extensions**

InputB Property

Description	Sets the Port Extension value for Receiver B
VB Syntax	*portExt*.**InputB** = *value*
Variable	**(Type) - Description**
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0
Examples	portExt.InputB = 10e-6 'Write inB = portExt.InputB 'Read
C++ Syntax	HRESULT get_InputB(double *pVal) HRESULT put_InputB(double newVal)
Interface	IPortExtension

Write/Read **About Port Extensions**

Port1 Property

Description	Sets a Port Extension value for Port 1
VB Syntax	*portExt*.**Port1** = *value*

Variable	**(Type) - Description**
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0
Examples	portExt.Port1 = 10e-6 'Write prt1 = portExt.Port1 'Read
C++ Syntax	HRESULT get_Port1(double *pVal) HRESULT put_Port1(double newVal)
Interface	IPortExtension

Write/Read **About Port Extensions**

Port2 Property

Description	Sets a Port Extension value for Port 2
VB Syntax	*portExt*.**Port2** = *value*
Variable	**(Type) - Description**
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0
Examples	portExt.Port2 = 10e-6 'Write prt2 = portExt.Port2 'Read
C++ Syntax	HRESULT get_Port2(double *pVal) HRESULT put_Port2(double newVal)
Interface	IPortExtension

Write/Read **About Port Extensions**

Port3 Property

Description	Sets a Port Extension value for Port 3
VB Syntax	*portExt*.**Port3** = *value*
Variable	**(Type) - Description**
portExt	A Port Extension **(object)**
value	**(double)** - Port Extension value in seconds. Choose any number between **-10** and **10**
Return Type	Double
Default	0
Examples	portExt.Port3 = 10e-6 'Write

	prt3 = portExt.Port3 'Read
C++ Syntax	HRESULT get_Port3(double *pVal) HRESULT put_Port3(double newVal)
Interface	IPortExtension

PowerLossSegments Collection

PowerLossSegments Collection

Description

A collection object that provides a mechanism for iterating through the segments of the power loss table used in source power calibration.

For more information, see Collections in the Analyzer.

Methods	**Description**
Add	Adds a PowerLossSegment object to the collection.
Item	Use to get a handle to a PowerLossSegment object in the collection.
Remove	Removes an object from the collection.
Properties	**Description**
Count	Returns the number of objects in the collection.
Parent	Returns a handle to the Parent object (SourcePowerCalibrator) of this collection.

Write-only **About Source Power Cal**

Add (PowerLossSegment) Method

Description	Adds a PowerLossSegment to the PowerLossSegments collection. To ensure predictable results, it is best to remove all segments before defining a new list of segments. For each segment in the collection, do a seg.Remove.
VB Syntax	*segs*.**Add** *(item [size])*
Variable	**(Type) - Description**
segs	**(object) -** A PowerLossSegments collection (object)
item	**(variant)** - Number of the new segment. If it already exists, a new segment is inserted at the requested position.
size	**(long integer) -** Optional argument. The number of segments to add, starting with item. If unspecified, value is set to 1.
Return Type	None
Default	Not Applicable
Examples	segs.Add 1, 4 'Adds segments 1,2,3 and 4
C++ Syntax	HRESULT Add(VARIANT index, long size);
Interface	IPowerLossSegments

PowerLossSegment Object
PowerLossSegment Object

Description

Contains the properties describing a segment of the power loss table used in source power calibration.

You can get a handle to one of these segments through the segments.Item Method of the PowerLossSegments collection.

Methods

None

Properties	**Description**
Frequency	The frequency (Hz) associated with this segment. Shared with the PowerSensorCalFactorSegment Object
Loss	The loss value (dB) associated with this segment.
SegmentNumber	Returns the number of this segment Shared with the PowerSensorCalFactorSegment Object

Write / Read **About Source Power Cal**

Frequency Property

Description	Sets or returns the frequency associated with a PowerSensorCalFactorSegment or Sets or returns the frequency associated with a PowerLossSegment.
VB Syntax	*object*.**Frequency** = *value*
Variable	**(Type) - Description**
object	**(object) -** PowerSensorCalFactorSegment or PowerLossSegment
value	**(double)** – Frequency in units of Hz. This can be any non-negative value (limited by the maximum value of double).
Return Type	Double
Default	0
Examples	seg.Frequency = 6e9 'Write freq = seg.Frequency 'Read
C++ Syntax	HRESULT put_Frequency(double newVal);

	HRESULT get_Frequency(double *pVal);
Interface	IPowerSensorCalFactorSegment IPowerLossSegment

Write / Read **About Source Power Cal**

Loss (Source Power Cal) Property

Description	Sets or returns the loss value associated with a PowerLossSegment.
VB Syntax	*lossSeg*.**Loss** = *value*
Variable	**(Type) - Description**
lossSeg	**(object) -** PowerLossSegment
value	**(double)** – Loss value in dB. This can be any value between 0 and 200.
Return Type	Double
Default	0
Examples	lossSeg.Loss = 0.5 'Write lossVal = lossSeg.Loss 'Read
C++ Syntax	HRESULT put_Loss(Double newVal); HRESULT get_Loss(Double *pVal);
Interface	IPowerLossSegment

Read-only **About Segment Sweep**

SegmentNumber Property

Description	Returns the number of the current segment, PowerSensorCalFactorSegment or PowerLossSegment object.
VB Syntax	*seg*.**SegmentNumber**
Variable	**(Type) - Description**
seg	**(object)** - A Segment, PowerSensorCalFactorSegment or PowerLossSegment. Get a handle to the object by referring to the item in the appropriate collection (Segments, CalFactorSegments or PowerLossSegments).
Return Type	Long Integer
Default	Not Applicable
Examples	segNum = seg.SegmentNumber 'returns the segment number -Read
C++ Syntax	HRESULT get_SegmentNumber(long *pVal)
Interface	ISegment IPowerSensorCalFactorSegment IPowerLossSegment

PowerSensor Object

Description

Each power sensor connected to the power meter associated with Source Power Calibration will have a PowerSensor object created to represent it. These PowerSensor objects reside in the PowerSensors collection within the SourcePowerCalibrator object. You cannot directly create PowerSensor objects, but can only retrieve existing ones from the PowerSensors collection.

The PowerSensorCalFactorSegment object is also accessed through the PowerSensor object. These are accessed through the CalFactorSegments collection in the PowerSensor object.

Example

```
Dim powerCalibrator as SourcePowerCalibrator
 Dim powerSensor as PowerSensor
 Dim calFactorSegment as PowerSensorCalFactorSegment

 Set powerCalibrator = pna.SourcePowerCalibrator

 ' Specify GPIB address of the power meter.
 powerCalibrator.PowerMeterGPIBAddress = 13

 ' Each time the PowerSensors collection is accessed, the power meter is
queried to determine which channels have sensors attached. The
collection is updated accordingly.

 If powerCalibrator.PowerSensors.Count > 0
 ' If channel B of the meter has a sensor attached but channel A does
not, then element 1 of the
 ' collection is sensor B.  Whenever channel A has a sensor, sensor A
will be element 1.
 Set powerSensor = powerCalibrator.PowerSensors(1)
 ' Insert one new PowerSensorCalFactorSegment at the beginning of the
collection (index 1).

 powerSensor.CalFactorSegments.Add(1)
 ' Assign our variable to refer to that object.
 Set calFactorSegment = powerSensor.CalFactorSegments(1)

 ' Set property values for that object.
 calFactorSegment.Frequency = 300000
 ' frequency in Hz
 calFactorSegment.CalFactor = 98
 ' cal factor in percent

 End If
```

Methods

None

Properties	**Description**
CalFactorsSegments (collection)	
MinimumFrequency	Minimum usable frequency (Hz) specified for this power sensor.
MaximumFrequency	Maximum usable frequency (Hz) specified for this power sensor.

PowerMeterChannel	Identifies which power sensor this object corresponds to (or which channel of the power meter the sensor is connected to).
ReferenceCalFactor	Reference cal factor (%) associated with this power sensor.

Write/Read **About Source Power Cal**

MaximumFrequency (Source Power Cal) Property

Description	Maximum usable frequency specified for the power sensor.
VB Syntax	*pwrSensor*.**MaximumFrequency** = *value*
Variable	**(Type) - Description**
pwrSensor	**(object) -** A PowerSensor (object)
value	**(double)** -Frequency in Hertz.
Return Type	Double
Default	0
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerSensors(1).MaximumFrequency = 6e9 'Write MaxFreq = powerCalibrator.PowerSensors(1).MaximumFrequency 'Read
C++ Syntax	HRESULT put_MaximumFrequency(double newVal); HRESULT get_MaximumFrequency(double *pVal);
Interface	IPowerSensor

Write/Read **About Source Power Cal**

MinimumFrequency (Source Power Cal) Property

Description	Minimum usable frequency specified for the power sensor.
VB Syntax	*pwrSensor*.**MinimumFrequency** = *value*
Variable	**(Type) - Description**
pwrSensor	**(object) -** A PowerSensor (object)
value	**(double)** -Frequency in Hertz.
Return Type	Double
Default	0
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerSensors(1).MinimumFrequency = 300e3 'Write MinFreq = powerCalibrator.PowerSensors(1).MinimumFrequency 'Read
C++ Syntax	HRESULT put_MinimumFrequency(double newVal); HRESULT get_MinimumFrequency(double *pVal);

Interface IPowerSensor

Read-only **About Source Power Cal**

PowerMeterChannel Property

Description Identifies which channel of the power meter the power sensor is connected to.

VB Syntax *chan* = *powerSensor*.**PowerMeterChannel**

Variable **(Type) - Description**

chan **(enum NAPowerAcquisitionDevice)** – Power meter channel identifier for sensor. Choose from:

0 – naPowerSensor_A

1 – naPowerSensor_B

pwrSensor **(object) -** A PowerSensor (object)

Return Type NAPowerAcquisitionDevice

Default Not Applicable

Examples

Set pwrCal = pna.SourcePowerCalibrator
meterChannel = pwrCal.PowerSensors(1).PowerMeterChannel

C++ Syntax HRESULT PowerMeterChannel(tagNAPowerAcquisitionDevice *pSensor);

Interface IPowerSensor

Read-only **About Source Power Cal**

ReferenceCalFactor Property

Description Reference cal factor (%) associated with this power sensor. This property and the CalFactorSegments collection are used to perform source power calibration **only** if the power sensor does not contain cal factors in EPROM (for example, HP/Agilent 848x sensors).

VB Syntax *powerSensor*.**ReferenceCalFactor** = *value*

Variable **(Type) - Description**

pwrSensor **(object) -** A PowerSensor (object)

value **(double)** – Cal factor in units of percent. This can be any value between 1 and 150.

Return Type Double

Default 100

Examples

```
Set powerCalibrator = pna.SourcePowerCalibrator
 powerCalibrator.PowerSensors(1).ReferenceCalFactor =
99 'Write
```

RefFact = powerCalibrator.PowerSensors(1).ReferenceCalFactor 'Read

---

C++ Syntax HRESULT put_ReferenceCalFactor(double newVal);

	HRESULT get_ReferenceCalFactor(double *pVal);
Interface	IPowerSensor

PowerSensorCalFactorSegment Object

PowerSensorCalFactorSegment Object

Description

Contains the properties describing a segment of a power sensor cal factor table.

You can get a handle to one of these segments through CalFactorSegments.Item(n)

Methods

None

Properties	**Description**
Frequency	The frequency (Hz) associated with this segment. Shared with the PowerLossSegment Object
CalFactor	The cal factor (%) associated with this segment.
SegmentNumber	Returns the number of this segment Shared with the PowerLossSegment Object

Write / Read	**About Source Power Cal**

CalFactor Property

Description	Sets or returns the cal factor value associated with a power sensor cal factor segment.
VB Syntax	*calFactSeg*.**CalFactor** = *value*
Variable	**(Type) - Description**
powerCalibrator	**(object) -** A PowerSensorCalFactorSegment (object)
value	**(double)** – Cal factor in percent. Choose any value between 1 and 150
Return Type	Double
Default	0
Examples	calFactSeg.CalFactor = 98 'Write factor = calFactSeg.CalFactor 'Read
C++ Syntax	HRESULT put_CalFactor(Double newVal); HRESULT get_CalFactor(Double *pVal);

Interface	IPowerSensorCalFactorSegment

PowerSensors Collection

PowerSensors Collection

Description

A collection object that provides a mechanism for iterating through the PowerSensor objects which are connected to the power meter. Each time this collection object is accessed, the power meter is queried to determine how many sensors are connected to it. The collection size and order of objects is then adjusted accordingly before the requested method or property operation is performed. The power meter is specified by using the PowerMeterGPIBAddress property of the SourcePowerCalibrator object.

For more information about collections, see Collections in the Analyzer.

Methods	**Description**
Item	Use to get a handle to a PowerSensor object in the collection.
Properties	**Description**
Count	Returns the number of objects in the collection.
Parent	Returns a handle to the Parent object (SourcePowerCalibrator) of this collection.

SCPIStringParser Object

SCPIStringParser Object **(default interface is ISCPIStringParser2)**

Default Interface is ISCPIStringParser2

Method	**Interface**	**Description**
Parse	ISCPIStringParser	Provides the ability to send a SCPI command from within the COM command.
Execute	**ISCPIStringParser2**	Does not convert scpi errors. Use :SYST:ERR?

Properties
None

Read-only

Execute Method

Description	This method can be used with SCPI command :SYST:ERR? to convert scpi errors into text.
VB Syntax	Scpi.**Execute**(SCPI_Command As String)
Variable	**(Type) - Description**
scpi	A ScpiStringParser **(Object)**
SCPI_Command	**(String)** - Any valid SCPI command
Return Type	String
Default	Not Applicable
Examples	**Dim scpi As ScpiStringParser** **Set scpi = app.ScpiStringParser** **scpi.Execute("SYST:PRES");** ErrorString = scpi.Execute("SYST:ERROr?");
C++ Syntax	Execute([in] BSTR SCPI_Command,[out,retval] BSTR * pQueryResponse);
Interface	IScpiStringParser2
Write-Read	**SCPI Command Tree**

Parse Method

Description	Executes a SCPI command.
VB Syntax	*scpi*.**Parse (***"SCPI command"***)**
Variable	**(Type) - Description**
scpi	A ScpiStringParser **(object)**
SCPI command	**(string)** - Any valid SCPI command
Return Type	String
Default	Not Applicable
Examples	Dim scpi As ScpiStringParser Set scpi = app.ScpiStringParser Dim startfreq As Double startfreq = 100e6 ' scpi.Parse ("Sens:Freq:Start " & startfreq)'Write Dim str As String str = scpi.Parse ("Sens:Freq:Start?")'Read
C++ Syntax	HRESULT Parse(BSTR SCPI_Command, BSTR *pQueryResponse)
Interface	IScpiStringParser

Segments Collection

Segments Collection

Description

A collection object that provides a mechanism for iterating through the sweep segments of a channel. Sweep segments are a potentially faster method of sweeping the analyzer through only the frequencies of interest. See Collections in the Analyzer

Methods	Description
Add	Adds an item to either the Segments collection.
Item	Use to get a handle to a segment in the collection..
Remove	Removes an item from a collection of objects.
Properties	**Description**
Count	Returns the number of items in a collection of objects.
IF Bandwidth Option	Enables the IFBandwidth to be set on individual sweep segments.
Parent	Returns a handle to the current naNetworkAnalyzer application..
Source Power Option	Enables setting the Source Power for a segment.

Write-only **About Segment Sweep**

Add (segment) Method

Description	Adds segments to the Segments collection, but does not turn the segments ON.
VB Syntax	*segs.***Add** *(item, [size])*
segs	A segments collection **(object)**
item	**(variant)** Number of the new segment. If it already exists, a new segment is inserted at the requested position.
size	**(long integer)** Optional argument. The number of segments to add, starting with *item*. If unspecified, value is set to 1.
Return Type	None
Default	None
Examples	Segs.Add 1, 4 'Adds segments 1,2,3,and 4. (does NOT automatically turn segments ON)
C++ Syntax	HRESULT Add(VARIANT index, long size);
Interface	ISegments
Remarks	To ensure predictable results, it is best to remove all segments before defining a segment list. For each segment in the collection, do a seg.Remove.

Write/Read **About Segment Sweep**

IFBandwidthOption Property

Description	Enables the IFBandwidth to be set on individual sweep segments. This property must be set True **before** *seg*.IFBandwidth = *value* is sent. Otherwise, this command will be ignored.
VB Syntax	*segs*.**IFBandwidthOption** = *value*
Variable	**(Type) - Description**
segs	A Segments collection **(object)**
value	**(boolean)**

	True - Enables variable IFBandwidth setting for segment sweep **False** - Disables variable IFBandwidth setting for segment sweep
Return Type	Boolean
Default	False
Examples	segs.IFBandwidthOption = True 'Write IFOption = IFBandwidthOption 'Read
C++ Syntax	HRESULT get_IFBandwidthOption(VARIANT_BOOL *pVal) HRESULT put_IFBandwidthOption(VARIANT_BOOL newVal)
Interface	ISegments

Write/Read **About Source Power**

SourcePowerOption Property

Description	Enables the source power to be set on individual sweep segments. This property must be set True **before** seg.TestPortPower = *value* is sent. Otherwise, the test port power command will be ignored.
VB Syntax	*segs*.**SourcePowerOption** = *state*
Variable	**(Type) - Description**
segs	A Segments collection **(object)**
state	**(boolean)** **True (1)** - Enables variable TestPortPower to be set segment sweep **False (0)** - Disables variable TestPortPower to be set segment sweep
Return Type	Boolean **True** - Enabled **False** - Disabled
Default	False
Examples	segs.SourcePowerOption = True 'Write powerOption = SourcePowerOption 'Read
C++ Syntax	HRESULT get_SourcePowerOption(VARIANT_BOOL *pVal) HRESULT put_SourcePowerOption(VARIANT_BOOL newVal)
Interface	ISegments

Segment Object

Segment Object (default interface is ISegment2)

Description

Contains the methods and properties that affect a sweep segment. You can get a handle to a sweep segment through the segments collection.[**segments.item(n)**.]

Note: All of these properties are shared with at least one of the following objects: Channel, PowerSensorCalFactorSegment or PowerLossSegment.

ISegments2 Interface extends the Segment interface with one Method and is the **default interface**.

Methods	**Interface**	**Description**
SetAllSegments	**ISegment2**	Uploads a segment table to the PNA.
Property	**Interface**	**Description**
centerFrequency	ISegment	Sets or returns the center frequency of the segment. Shared with the Channel Object
DwellTime	ISegment	Dwell time value. Shared with the Channel Object
FrequencySpan	ISegment	Sets or returns the frequency span of the segment. Shared with the Channel Object
IFBandwidth	ISegment	Sets or returns the IF Bandwidth of the segment. Shared with the Channel Object
NumberOfPoints	ISegment	Sets or returns the Number of Points of the segment. Shared with the Channel Object
Segment Number	ISegment	Returns the number of the current segment.
StartFrequency	ISegment	Sets or returns the start frequency of the segment. Shared with the Channel Object
State	ISegment	Turns On or OFF a segment.
StopFrequency	ISegment	Sets or returns the stop frequency of the segment. Shared with the Channel Object
TestPortPower	ISegment	Sets or returns the RF power level of the segment. Shared with the Channel Object

Write-only **About Segment Sweep**

SetAllSegments Method

Description	Uploads a segment table to the PNA replacing any existing segment table. Segments must be ascending in frequency and non-overlapping. If they are not, the segments are 'adjusted' as they are from the front panel control. The total number of points for all segments cannot exceed the PNA maximum for a sweep. See an example that creates a 2-dimensional array of Doubles of 7 x numSegs+1 that contains the segment data. You can see from the comments the order in which the segment elements are specified: index 0 is segment state, index 4 is IFBW, and so forth.
VB Syntax	*Seg*.**SetAllSegments** (*segdata*)
Variable	**(Type) - Description**
seg	A Segment **(Object)**

segdata	Variant or Double Array - Segment data For VARIANT, the underlying type must be appropriate for the element: Boolean - segment on/off Integer - number of points Double - all other elements.
Return Type	Not Applicable
Default	Not Applicable
Examples	See an example using this command
C++ Syntax	SetAllSegments (VARIANT Segments);
Interface	ISegments2

SourcePowerCalibrator Object

SourcePowerCalibrator Object **(Default interface is ISourcePowerCalibrator)**

Description

This object is a child object of Application, and is a vehicle for performing source power calibrations.

Method	**Description**
AbortPowerAcquisition	Aborts a source power cal acquisition sweep that is currently in progress.
AcquirePowerReadings	Initiates a source power cal acquisition.
ApplyPowerCorrectionValues	Applies correction values after completing a source power cal acquisition sweep.
SetCalInfo	Specifies the type of source power calibration about to be performed, and instrument state-related settings for which it is to be performed.
Property	**Description**
CalPower	Specifies the power level that is expected at the desired reference plane (input or output of the device-under-test).
PowerLossSegments (collection)	
PowerMeterGPIBAddress	Specifies the GPIB address of the power meter that will be referenced by this SourcePowerCalibrator object.
PowerSensors (collection)	
ReadingsPerPoint	For purpose of averaging, specifies how many power readings are taken at each frequency point (Averaging factor).
UsePowerLossSegments	Specifies if subsequent calls to the AcquirePowerReadings method will make use of the loss table (PowerLossSegments).
UsePowerSensorFrequencyLimits	Specifies if subsequent calls to the AcquirePowerReadings method will make use of power sensor frequency checking capability.

Write-only **About Source Power Cal**

AbortPowerAcquisition Method

Description Aborts a source power cal acquisition sweep that is currently in progress.

VB Syntax *powerCalibrator*.**AbortPowerAcquisition**

Variable	**(Type) - Description**
powerCalibrator	**(object)** - A SourcePowerCalibrator object
Return Type	None
Default	Not Applicable

Examples powerCalibrator.AbortPowerAcquisition

C++ Syntax HRESULT AbortPowerAcquisition();

Interface ISourcePowerCalibrator

Write-only **About Source Power Cal**

AcquirePowerReadings Method

Description Initiates a source power cal acquisition.

VB Syntax *powerCalibrator*.**AcquirePowerReadings** *device [,sync]*

Variable	**(Type) - Description**
powerCalibrator	**(object)** - A SourcePowerCalibrator object
device	**(enum NAPowerAcquisitionDevice)** The specific device (sensor on the power meter) to be used for the acquisition. Choose from: **0 – naPowerSensor_A** **1 – naPowerSensor_B**
sync	**(boolean)** Optional argument. If not specified, value is set to False. Choose from: **True (1)** – The method does not return until this acquisition has completed (the program calling this method is halted while waiting for the method to return). **False (0)** – The method initiates an acquisition then returns immediately (while the acquisition still proceeds). The program calling this method can then perform other operations during the acquisition.
Return Type	None
Default	Not Applicable

Examples powerCalibrator.AcquirePowerReadings naPowerSensor_A, True

C++ Syntax HRESULT AcquirePowerReadings(tagNAPowerAcquisitionDevice enumAcqDevice, VARIANT_BOOL bSync);

Interface ISourcePowerCalibrator

Write-only **About Source Power Cal**

ApplyPowerCorrectionValues Method

Description Applies the array of power correction values to the channel memory and turns correction ON. Perform after completing a source power cal acquisition sweep or after programmatic input of source power correction values (see putSourcePowerCalData Method and putSourcePowerCalDataScalar Method). If using these methods, correction is not turned ON if the current number of points on the channel is not equal to the number of values that were input.

VB Syntax *powerCalibrator*.ApplyPowerCorrectionValues

Variable	**(Type) - Description**
powerCalibrator	**(object)** - A SourcePowerCalibrator object
Return Type	None
Default	Not Applicable

Examples powerCalibrator.ApplyPowerCorrectionValues

C++ Syntax HRESULT ApplyPowerCorrectionValues();

Interface ISourcePowerCalibrator

Write-only **About Source Power Cal**

SetCalInfo Method (for source power cals)

Description Specifies the technique to be used for the source power calibration about to be performed, and the channel and source port for which it is to be performed.

VB Syntax *powerCalibrator*.**SetCalInfo** *calMethod, channel, sourcePort, calPower*

Variable	**(Type) - Description**
powerCalibrator	**(object)** - A SourcePowerCalibrator object
calMethod	**(enum NASourcePowerCalMethod)** The method of gathering the source power correction data. **0 – naPowerMeter** (the only method currently supported)
channel	**(long integer) -** Number of the PNA channel (not power meter channel) on which the source power cal will be performed. If the channel doesn't already exist, it will be created.
sourcePort	**(long integer) -** Port number on which the source power cal will be performed.
calPower	**(double) -** Specifies the power level that is expected at the desired reference plane (input or output of DUT) following the source power cal.
Return Type	None
Default	Not Applicable

Examples powerCalibrator.SetCalInfo naPowerMeter, 1, 1, -10

C++ Syntax HRESULT SetCalInfo(tagNASourcePowerCalMethod calMethod, long channel, long sourcePort, double calPower);

Interface ISourcePowerCalibrator

Read-only **About Source Power Cal**

CalPower Property

Description Specifies the power level that is expected at the desired reference plane

	(input or output of the device-under-test) following a source power calibration.
VB Syntax	*value = powerCalibrator*.**CalPower** *(chan, sourcePort)*
Variable	**(Type) - Description**
value	**(double)** - Variable to store the returned Cal power value in dBm.
powerCalibrator	**(object)** - A SourcePowerCalibrator object
chan	**(long integer)** - Channel number of the PNA.
sourcePort	**(long integer)** - Source port number
Return Type	None
Default	0
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.CalPower = -10 'Write power = powerCalibrator.CalPower 'Read
C++ Syntax	HRESULT get_CalPower(long channel, long sourcePort, double *pVal);
Interface	ISourcePowerCalibrator

Write / Read **About Source Power Cal**

PowerMeterGPIBAddress Property

Description	Specifies the GPIB address of the power meter that will be referenced by the SourcePowerCalibrator object.
VB Syntax	*powerCalibrator*.**PowerMeterGPIBAddress** = *value*
Variable	**(Type) - Description**
powerCalibrator	**(object) -** A SourcePowerCalibrator (object)
value	**(long integer)** – Power meter GPIB address. Choose any number between 0 and 30.
Return Type	Long integer
Default	13
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerMeterGPIBAddress = 13 'Write pwrMtrAddress = powerCalibrator.PowerMeterGPIBAddress 'Read
C++ Syntax	HRESULT put_PowerMeterGPIBAddress(long newVal); HRESULT get_PowerMeterGPIBAddress(long *pVal);
Interface	ISourcePowerCalibrator

Write / Read **About Source Power Cal**

ReadingsPerPoint Property

Description	For purpose of averaging during source power cal, specifies how many power

	readings are taken at each frequency point (Averaging factor).
VB Syntax	*pwrCal*.**ReadingsPerPoint** = *value*
Variable	**(Type) - Description**
pwrCal	**(object) -** A SourcePowerCalibrator (object)
value	**(long integer)** – Number of power readings. Choose any number between 1 and 100.
Return Type	Long Integer
Default	1
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.ReadingsPerPoint = 2 'Write numReadings = powerCalibrator.ReadingsPerPoint 'Read
C++ Syntax	HRESULT put_ReadingsPerPoint(long newVal); HRESULT get_ReadingsPerPoint(long *pVal);
Interface	ISourcePowerCalibrator

Write / Read **About Source Power Cal**

UsePowerLossSegments Property

Description	Specifies if subsequent calls to the AcquirePowerReadings method will make use of the loss table (PowerLossSegments).
VB Syntax	*pwrCal*.**UsePowerLossSegments** = *value*
Variable	**(Type) - Description**
pwrCal	**(object)** – A SourcePowerCalibrator (object)
value	**(boolean) -** **False (0)** – Do not use loss table **True (1)** – Use loss table
Return Type	Boolean
Default	False
Examples	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.UsePowerLossSegments = 1 'Write lossTableState = powerCalibrator.UsePowerLossSegments 'Read
C++ Syntax	HRESULT put_UsePowerLossSegments(VARIANT_BOOL bState); HRESULT get_UsePowerLossSegments(VARIANT_BOOL *bState);
Interface	ISourcePowerCalibrator

Write / Read **About Source Power Cal**

UsePowerSensorFrequencyLimits Property

Description	Specifies if subsequent calls to the AcquirePowerReadings method will observe frequency values of the MinimumFrequency and

	MaximumFrequency properties.
VB Syntax	*pwrCal*.**UsePowerSensorFrequencyLimits** = *value*

Variable	**(Type) - Description**
pwrCal	**(object)** – A SourcePowerCalibrator (object)
value	**(boolean) -** **False (0)** – Do not use power sensor frequency limits. An acquisition will use just one power sensor for the entire sweep, regardless of frequency. **True (1)** – Use power sensor frequency limits. A requested acquisition will only succeed for those frequency points which fall between the MinimumFrequency and MaximumFrequency values of that PowerSensor. An acquisition will pause in mid-sweep if the frequency is about to exceed the MaximumFrequency value. When the sweep is paused in this manner, a sensor connected to the other channel input of the power meter can be connected to the measurement port in place of the previous sensor, and then the sweep completed by another call to AcquirePowerReadings. However, the MaximumFrequency specified for the second sensor would need to be sufficient for the sweep to complete.
Return Type	Boolean
Default	False (0)

Examples

```
Set powerCalibrator = pna.SourcePowerCalibrator
 powerCalibrator.UsePowerSensorFrequencyLimits = 1 'Write
 FreqCheck = powerCalibrator.UsePowerSensorFrequencyLimits 'Read
```

C++ Syntax

```
HRESULT put_UsePowerSensorFrequencyLimits(VARIANT_BOOL bState);
HRESULT get_UsePowerSensorFrequencyLimits(VARIANT_BOOL *bState);
```

Interface ISourcePowerCalibrator

Trace Object

Trace Object (default interface is ITrace)

Description

The Trace object takes measurement data and control how the data is painted on the display. You can control scale, reference position, and reference line from the Trace Object.

Methods	**Description**
Autoscale	Autoscales the trace or all of the traces in the selected window. Shared with the NAWindow Object

Property	**Description**
Name	Sets or returns the trace name
ReferencePosition	Sets or returns the Reference Position of the active trace.
ReferenceValue	Sets or returns the value of the Y-axis Reference Level of the active trace.
YScale	Sets or returns the Y-axis Per-Division value of the active trace.

Write/Read **About Traces**

Name (trace) Property

Description

Sets or returns the name of the Trace. Use the trace name to identify the trace and refer to the trace in the collection.

Note: This is the same name as meas.Name; when one changes, the other changes.

VB Syntax *trac*.**Name** = *value*

Variable	**(Type) - Description**
trac	A Trace **(object)**
value	**(String)** Trace name
Return Type	String
Default	"CH1_S11_1" - name of the default measurement

Examples

trace.Name = "myTrace" 'Write
traceName = Name.Trace 'Read

C++ Syntax

HRESULT put_Name(BSTR name)
HRESULT get_Name(BSTR *name)

Interface ITrace

Write/Read **About Reference Position**

ReferencePosition Property

Description Sets or returns the Reference Position of the active trace.

VB Syntax *trce*.**ReferencePosition** = *value*

Variable	**(Type) - Description**
trce	A Trace **(object)**
value	**(double)** - Reference position on the screen measured in horizontal graticules from the bottom of the screen. Choose from any number between: **0** and **10.**
Return Type	Double
Default	0

Examples

meas.ReferencePosition = 5 'Middle of the screen -Write
rpos = meas.ReferencePosition -Read

C++ Syntax

HRESULT get_ReferencePosition(double *pVal)
HRESULT put_ReferencePosition(double newVal)

Interface ITrace

Write/Read **About Reference Level**

ReferenceValue Property

Description	Sets or returns the value of the Y-axis Reference Level of the active trace.
VB Syntax	*trce*.**ReferenceValue** = *value*
Variable	**(Type) - Description**
trce	A Trace **(object)**
value	**(double)** - Reference Value. Units and range depend on the current data format.
Return Type	Double
Default	Not applicable
Examples	meas.ReferenceValue = 0 'Write rlev = meas.ReferenceValue 'Read
C++ Syntax	HRESULT get_ReferenceValue(double *pVal) HRESULT put_ReferenceValue(double newVal)
Interface	ITrace

Write/Read **About Scale**

YScale Property

Description	Sets or returns the Y-axis Per-Division value of the active trace.
VB Syntax	*trace*.**YScale** = *value*
Variable	**(Type) - Description**
trace	A Trace **(object)**
value	**(double)** - Scale /division number. Units and range depend on the current data format.
Return Type	Double
Default	10 (db)
Examples	trac.YScale = 5 'Write yscl = trac.YScale 'Read
C++ Syntax	HRESULT get_YScale(double *pVal) HRESULT put_YScale(double newVal)
Interface	ITrace

Traces Collection

Traces Collection

Description

Child of the **Application** Object. A collection that provides a mechanism for getting a handle to a trace or iterating through the traces in a window.

Methods	**Description**
Item	Use to get a handle to a trace
Properties	**Description**
Count	Returns the number of traces in the collection.
Parent	Returns a handle to the current Application.

Transform Object

Transform Object (default interface is ITransform)

Description

Contains the methods and properties that control Time Domain transforms.

Methods	**Description**
SetFrequencyLowPass	Sets low frequencies for low pass.
Property	**Description**
Center	Sets or returns the Center time. Shared with the Gating Object
ImpulseWidth	Sets or returns the Impulse Width of Time Domain transform windows.
KaiserBeta	Sets or returns the Kaiser Beta of Time Domain transform windows.
Mode	Sets the type of transform.
Span	Sets or returns the Span time. Shared with the Gating Object
Start	Sets or returns the Start time. Shared with the Gating Object
State	Turns an Object ON and OFF.
StepRiseTime	Sets or returns the Rise time of the stimulus in Low Pass Step Mode.
Stop	Sets or returns the Stop time. Shared with the Gating Object

Write-only **About Time Domain**

SetFrequencyLowPass Method

Description	Set the start frequencies when **trans.Mode = LowPass**.
VB Syntax	*trans*.**SetFrequencyLowPass**

Variable	**(Type) - Description**
trans	A Transform **(object)**
Return Type	Not Applicable

Default	Not Applicable
Examples	trans.SetFrequencyLowPass
C++ Syntax	HRESULT SetFrequencyLowPass(void)
Interface	ITransform

Write/Read **About Time Domain**

ImpulseWidth Property

Description	Sets or returns the Impulse Width of Time Domain transform windows
VB Syntax	*trans*.**ImpulseWidth** = *value*
Variable	**(Type) - Description**
trans	A Transform **(object)**
value	**(double)** - Impulse Width in seconds. Range of settings depends on the frequency range of your analyzer.
Return Type	Double
Default	.98 / Default Span
Examples	trans.ImpulseWidth = 200e-12 'sets the Impulse width of a transform window - Write IW = trans.ImpulseWidth 'Read
C++ Syntax	HRESULT get_ImpulseWidth(double *pVal) HRESULT put_ImpulseWidth(double newVal)
Interface	ITransform

Write/Read **About Time Domain**

KaiserBeta Property

Description	Sets or returns the Kaiser Beta of Time Domain transform windows
VB Syntax	*trans*.**KaiserBeta** = *value*
Variable	**(Type) - Description**
trans	A Transform **(object)**
value	**(single)** - Kaiser Beta. Choose any number between **0** and **13.**
Return Type	Single
Default	0
Examples	trans.KaiserBeta = 6 'sets the Kaiser Beta of a transform window -Write KB = trans.KaiserBeta 'Read
C++ Syntax	HRESULT get_KaiserBeta(float *pVal) HRESULT put_KaiserBeta(float newVal)
Interface	ITransform

Write/Read **About Time Domain**

Mode Property

Description	Sets the type of transform.
VB Syntax	*trans*.**Mode** = *value*
Variable	**(Type) - Description**
trans	A Transform **(object)**
value	**(enum NATransformMode)** - Choose from: **0** - naTransformBandpassImpulse **1** - naTransformLowpassImpulse **2** - naTransformLowpassStep
Return Type	NATransformMode
Default	**0** - naTransformBandpassImpulse
Examples	trans.Mode = naTransformLowpassStep 'Write transmode = trans.Mode 'Read
C++ Syntax	HRESULT get_Mode(tagNATransformMode *pVal) HRESULT put_Mode(tagNATransformMode newVal)
Interface	ITransform

Write/Read **About Time Domain**

StepRiseTime Property

Description	Sets or returns the Rise time of the stimulus in Low Pass Step Mode.
VB Syntax	*trans*.**StepRiseTime** = *value*
Variable	**(Type) - Description**
trans	A Transform **(object)**
value	**(double)** - Rise time in seconds. Choose any number between **5.0e-13** and **1.63e-14.**
Return Type	Double
Default	0
Examples	trans.StepRiseTime = 1.0e-14 'sets the step rise time to 100 psec. -Write rt = trans.StepRiseTime 'Read
C++ Syntax	HRESULT get_StepRiseTime(double *pVal) HRESULT put_StepRiseTime(double newVal)
Interface	ITransform

COM Examples

Agilent VEE Example

Application Configuration

For this example use Agilent VEE version 6.0 or above which contains the Variant data type used to transfer data from the PNA. The type library for the PNA should be referenced in the Agilent VEE development environment.

Using the Agilent VEE Object Browser the developer can see the classes and methods which are available for development of applications for the PNA Series analyzer.

Application Code

There is a runtime version of Agilent VEE that may be used if the application has been saved as "runtime". A free version of Agilent VEE can be found on the following web site: http://www.agilent.com/find/vee/. The application may be run on a PC or on the PNA Series analyzer.

The application file is located at http://agilent.com/find/pna_applications.

C++ Example

The following example uses the smart pointer created by Microsoft Visual Studio. The calls to CoInitialize and CoUninitialize open and close the COM libraries.

Also notice that the pointers local to the main routine are explicitly released. When smart pointers go out of scope, they will perform this duty implicitly. However, we are calling CoUninitialize before they have the chance to be destroyed, so we are obliged to release them.

```
// An example program to illustrate the use of #import to bind to the
 // PNA type library.
 //
 #include "stdafx.h"
 #include "stdio.h"
 #include "math.h"

 ///////////////////////////////////////////////
 // import the network analyzer type library
 ///////////////////////////////////////////////
 #import "C:\Program Files\Common Files\Agilent\Pna\835x.tlb"
no_namespace, named_guids
 ///////////////////////////////////////////////
 // include the error definitions for the PNA so we can implement
 // error handling.
 ///////////////////////////////////////////////
 #include "C:\Program Files\Common
Files\Agilent\Pna\errorsystemmessage.h"

 IApplicationPtr pNA; // top level application pointer
 float fScalarData [1601]; // global buffer for data retrieval
 float fScalarData2[1601];

 DWORD dwCookie;

 ///////////////////////////////////////////////
 // SetupChannel:
 //
 // input: pointer to the channel
 //
 // function: sets properties on the channel
 ///////////////////////////////////////////////
 void SetupChannel(IChannelPtr pChannel)
 {
   pChannel->put_StartFrequency( 1.2E9 );
   pChannel->put_StopFrequency ( 4.2E9 );
```

```
   pChannel->put_NumberOfPoints ( 201);

 }

 ////////////////////////////////////////////
 // AcquireData:
 //
 // input: pointer to the channel
 //
 // function: single sweeps the channel
 ////////////////////////////////////////////
 void AcquireData( IChannelPtr pChannel )
 {
   pChannel->Single( TRUE );
 }

 ////////////////////////////////////////////
 // ReadData:
 //
 // input: pointer to the Measurement object
 //
 // function: reads data from the measurment's formatted
 // result data buffer
 ////////////////////////////////////////////
 void ReadScalarData(IMeasurementPtr pMeas )
 {
   IArrayTransferPtr pDataTransfer;
   pDataTransfer = pMeas;
   long numVals = 1601;
   float* pData = fScalarData;

   if(pDataTransfer){

   pDataTransfer->getScalar( naMeasResult, naDataFormat_LogMag,
&numVals, pData);

   for (int i = 0; i < numVals; i++)
   printf("%d\t%f\n",i,pData[i]);
   }
   TCHAR msg[100];
   BSTR param;
   pMeas->get_Parameter(&param);
   swprintf(msg,L"Review %s data",param);
   MessageBox(NULL,msg,L"User Message",0);
   ::SysFreeString(param);
 }


 void ReadComplexData(IMeasurementPtr pMeas )
 {
   IArrayTransferPtr pDataTransfer;
   pDataTransfer = pMeas;
   long numVals = 1601;
   float* pReal= fScalarData;
   float* pImag = fScalarData2;

   if(pDataTransfer){

   pDataTransfer->getPairedData( naRawData, naRealImaginary, &numVals,
pReal, pImag);
```

```
   for (int i = 0; i < numVals; i++)
   printf("%d\t%f\t%f\n",i,pReal[i], pImag[i]);
   }
   TCHAR msg[100];
   BSTR param;
   pMeas->get_Parameter(&param);
   swprintf(msg,L"Review %s data",param);
   MessageBox(NULL,msg,L"User Message",0);
   ::SysFreeString(param);
 }
 ///////////////////////////////////////////
 // PutData:
 //
 // input: pointer to the Measurement object
 //
 // function: writes data to the measurment's raw data
 // buffer
 ///////////////////////////////////////////
 void PutData( IMeasurementPtr pMeas )
 {
   IArrayTransferPtr pDataTransfer;
   pDataTransfer = pMeas;
   long numVals = 201;

   if(pDataTransfer){

   NAComplex* pComplex = new NAComplex[numVals];

   pComplex[0].Im = 0;
   pComplex[0].Re = 1;
   for (int i = 1; i < numVals; i++)
   {
   pComplex[i].Im = (float)sin(i)/i;
   pComplex[i].Re = (float)cos(i)/i;
   }

   pDataTransfer->putNAComplex( naRawData, numVals, pComplex,
naDataFormat_Polar);
   delete [] pComplex;
   }
 }

 ///////////////////////////////////////////
 // printError
 ///////////////////////////////////////////
 void printError( HRESULT hr)
 {
   BSTR text;

   hr = pNA->get_MessageText ((NAEventID) hr, &text);
   MessageBox(NULL,text,L"Network Analyzer error",0);
   ::SysFreeString(text);
 }


 ///////////////////////////////////////////
 // main
 ///////////////////////////////////////////
 int main(int argc, char* argv[])
 {
```

```
    HRESULT hr;
    const long channel1 = 1;
    const long window1 = 1;
    const long srcport = 1;
    IMeasurementPtr pMeasurement;
    IChannelPtr pChannel;

    // initialize COM libraries
    CoInitialize(NULL);

    try {
    pNA = IApplicationPtr("AgilentPNA835x.Application.1");

    pNA->put_Visible(TRUE);
    pNA->Reset();


    pNA->CreateMeasurement (channel1, "S21",srcport, 5);
    hr = pNA->get_ActiveChannel( &pChannel);

    if (SUCCEEDED (hr))
    {
    SetupChannel( pChannel);
    AcquireData(pChannel);
    }

    hr= pNA->get_ActiveMeasurement( &pMeasurement);
    if (SUCCEEDED(hr))
    {
    pMeasurement->put_Format( naDataFormat_Polar);
    ReadScalarData( pMeasurement);
    ReadComplexData( pMeasurement);
    PutData(pMeasurement);
    }
    if (FAILED(hr))
    {
    printError(hr);
    }

    // make sure to release the remaining pointers
    // before calling CoUninitialize

    pMeasurement.Release();
    pChannel.Release();
    pNA.Release();
    }
    catch (_com_error err)
    {
    printError( err.Error() );
    }

    CoUninitialize();
    return 0;
  }
```

ECAL Confidence Check

This Visual Basic program:

- Initializes the PNA objects.
- Performs a complete ECAL confidence check

Before using this code:

- The active channel must contain an S11 measurement with a 1-port or N-port calibration
- Prepare a form with two buttons named **cmdRun** and **cmdQuit**

```
Private oPNA As AgilentPNA835x.Application
Private oChan As Channel
Private oCal As Calibrator
Private oMeas As Measurement

Private Sub cmdRun_Click()
Dim iMeasIndex As Integer

Set oPNA = CreateObject("AgilentPNA835x.Application", "MachineName")
Set oChan = oPNA.ActiveChannel
Set oCal = oChan.Calibrator

iMeasIndex = 1

' Loop through measurements until an S11 on the active channel
' is found, or the end of the measurement collection is reached.
Do
   Set oMeas = oPNA.Measurements(iMeasIndex)
   If oMeas.Parameter = "S11" And _
      oMeas.channelNumber = oChan.channelNumber Then Exit Do
   iMeasIndex = iMeasIndex + 1
   If iMeasIndex > oPNA.Measurements.Count Then
      MsgBox "No S11 measurement found on the active channel." _
             & " Create an S11 measurement, then try again."
      Exit Sub
   End If
Loop

' Set up trace view so we are viewing only the data trace.
oMeas.View = naData
' Acquire the S11 confidence check data from ECal Module A
' into the memory buffer.
oCal.AcquireCalConfidenceCheckECAL "S11", naECALModule_A
' Turn on trace math so the trace shows data divided by memory.
```

```
' You can be confident the S11 calibration is reasonably good if
' the displayed trace varies no more than a few tenths of a dB
' from 0 dB across the entire span.
oMeas.TraceMath = naDataDivMemory
End Sub

Sub cmdQuit_Click()
' Turn off trace math
' in case someone clicks Quit without having clicked Run
If oMeas <> Nothing Then oMeas.TraceMath = naDataNormal
' Conclude the confidence check to set the ECal module
' back to it's idle state.
If oCal <> Nothing Then oCal.DoneCalConfidenceCheckECAL
' End the program
   End
End Sub
```


COM CalSet program examples using C++

- Get example, cycles through the calsets collection, printing values from each error term buffer.
- Put example, creates a calset and a buffer using SafeArrayCreate.
- Put and Get example, creates a calset, writes a buffer to it and reads it back.

Get example

This example cycles through the calsets collection, printing values from each error term buffer.

The example uses the methods **GetErrorTermList2 and GetErrorTermByString**. The methods **GetStandardsList2 and GetStandardByString** are used similarly.

```
#include "stdafx.h"
#include "atlbase.h"
using namespace std;
#import "C:\Program Files\Agilent\Network Analyzer\835x.tlb"
raw_interfaces_only, named_guids, no_namespace
inline void HR( HRESULT hr)
{
if (FAILED(hr))
throw hr;
}
int main()
{
CoInitialize(NULL);
CComPtr spPNA;
if (FAILED(CoCreateInstance(CLSID_Application, NULL, CLSCTX_SERVER,
IID_IApplication, (void**)&spPNA)))
{
MessageBox(NULL, "could not create PNA","",0);
return 1;
}
try {
```

```
long setCount;
CComBSTR unfiltered("");
// ** initialize interface handlesCComPtr spMgr;
HR(spPNA->GetCalManager(&spMgr ));
CComPtr spSets;
HR(spMgr->get_CalSets( &spSets) );   // Get the calset collection
HR(spSets->get_Count( &setCount));
// ** loop through the collectionfor (int i = 1; i  ));
HR(spSet->QueryInterface( &spSet2 ));

VARIANT buflist;
HR(spSet2->GetErrorTermList2(0, unfiltered, &buflist));

// ** loop through all the error term buffers in the calsetVARIANT*
pvStrings;
HR(SafeArrayAccessData( buflist.parray, (void**)&pvStrings));
for ( int bufNum = 0; bufNum rgsabound[0].cElements; bufNum++ )
{
VARIANT vOut;
BSTR bufName = pvStrings[bufNum].bstrVal;
HR(spSet2->GetErrorTermByString( 0, bufName, &vOut));
cout get_MessageText((enum NAEventID) hr, &bstrMsg );
}
spPNA.Release();
CoUninitialize();
return 0;
}
```

Put example

This example creates a calset and a buffer using SafeArrayCreate.

The example uses the methods **PutErrorTermByString** method to put the buffer in the calset. The **PutStandardByString** is a similarly used method.

```
  // PutErrorTermByString.cpp : Defines the entry point for the console
application.
//#include "stdafx.h"
#include "atlbase.h"

#import "C:\Program Files\Agilent\Network Analyzer\835x.tlb"
raw_interfaces_only, named_guids, no_namespace
inline void HR( HRESULT hr)
{
if (FAILED(hr))
throw hr;
}
using namespace std;
int main()
{
CoInitialize(NULL);
CComPtr spPNA;
if (FAILED(CoCreateInstance(CLSID_Application, NULL, CLSCTX_SERVER,
IID_IApplication, (void**)&spPNA)))
{
MessageBox(NULL, "could not create PNA","",0);
return 1;
}
try {
HR(spPNA->Preset());

// generate a safearray of floats
```

```
SAFEARRAYBOUND bounds[2];
 bounds[0].cElements = 201;
bounds[0].lLbound = 0;
bounds[1].cElements = 2;
bounds[1].lLbound = 0;
float realPart = 1.0;
float imagPart = 0.0;
long indices[2];
long maxPts = 201;
SAFEARRAY* psa = SafeArrayCreate( VT_R4, 2, bounds );
for (int pt = 0 ; pt GetCalManager(&spMgr ));
HR(spMgr->CreateCalSet(1, &spSet));
spSet2 = spSet;
CComBSTR bufName("MyPhonyCalSet:MyPhonyBuffer");
HR(spSet2->PutErrorTermByString( bufName, complexData));
HR(spSet2->Save());
}
catch (HRESULT hr)
{
CComBSTR bstrMsg;
spPNA->get_MessageText((enum NAEventID) hr, &bstrMsg );
MessageBox( NULL, (LPCTSTR)CString(bstrMsg), "Error",MB_OK);
}
spPNA.Release();
CoUninitialize();
return 0;
```

Put and Get example for ICalData3

This example creates a calset, writes a buffer to it and reads it back. The example uses the methods **PutErrorTermComlexByString** and **GetErrorTermComlexByString** methods. The **PutStandardComplexByString** and **GetStandardComplexByString** methods are used similarly.

```
#include "stdafx.h"
#include "atlbase.h"
#include #include

#import "C:\Program Files\Agilent\Network Analyzer\835x.tlb"
raw_interfaces_only, named_guids, no_namespace
inline void HR( HRESULT hr)
{
if (FAILED(hr))
throw hr;
}
using namespace std;
int main()
{
CoInitialize(NULL);
CComPtr spPNA;
if (FAILED(CoCreateInstance(CLSID_Application, NULL, CLSCTX_SERVER,
IID_IApplication, (void**)&spPNA)))
{
MessageBox(NULL, "could not create PNA","",0);
return 1;
}
try {
HR(spPNA->Preset());

// generate some data for our calset bufferstd::vector real(201,0);
std::vector imag(201,0);
for (int i = 0; i ;
CComPtr ;
CComQIPtr ;
// Create a calset
```

```
HR(spPNA->GetCalManager(&spMgr ));
HR(spMgr->CreateCalSet(1, &spSet));
spCalData3 = spSet;
// insert a bufferCComBSTR bufName("Example Cal Set:Bogus Data Buffer");
HR(spCalData3->PutErrorTermComplexByString( bufName, real.size(),
&real[0], &imag[0]));
HR(spSet->Save());
// read the buffer back outlong pts = real.size();
real.assign(pts,0);
imag.assign(pts,0);
HR(spCalData3->GetErrorTermComplexByString( 0, bufName, &pts, &real[0],
&imag[0]));
}
catch (HRESULT hr)
{

CComBSTR bstrMsg;
spPNA->get_MessageText((enum NAEventID) hr, &bstrMsg );
MessageBox( NULL, (char*)_bstr_t(bstrMsg.m_str), "Error",MB_OK);
}
spPNA.Release();
CoUninitialize();
return 0;
}
```

COM Events Example

This Visual Basic program shows how to monitor the end of sweep.

The program will set sweep time to various amounts and BEEPs when sweep is completed. This method allows other processes to continue while waiting for end-of-sweep. This program stops after 10 loops.

Note: To avoid **Permission Denied** problems, this should be run on the PNA and not a PC. To run it from a PC both units must be "trusted" and on the same domain/workgroup.

```
Option Explicit
 Dim na As AgilentPNA835x.Application
 Dim WithEvents naEvnt As AgilentPNA835x.Application
 Dim ch As AgilentPNA835x.Channel
 Dim sweepComplete As Boolean

 Private Sub Form_Load()

 Dim N As Integer
 Set na = CreateObject("AgilentPNA835x.application")
 na.preset
 Set ch = na.ActiveChannel
 na.DisallowAllEvents            ' Turn off all events
 Set naEvnt = na                 ' Enable event interrupts
 Do
 N = N + 1                     ' Loop counter
 ch.sweepTime = 1 + (Rnd * 9)  ' Set random sweep-time from 1-10 sec
 sweepComplete = False
 ch.Single False               ' Trigger sweep
 naEvnt.AllowEventCategory naEventCategory_CHANNEL, True  ' Enable
Channel event
```

```
 Do
  DoEvents                    ' Allows other processes to continue
 Loop Until sweepComplete = True
 naEvnt.AllowEventCategory naEventCategory_CHANNEL, False  ' Disable
event until ready for next one
 Beep                          ' Do end-of-sweep processing here;

 Loop Until N > 10
 End

 End Sub

 Private Sub naEvnt_OnChannelEvent(ByVal eventID As Variant, ByVal
chNumber As Variant)
 ' In this example we don't care about the channel info
 If eventID = naEventID_CHANNEL_TRIGGER_COMPLETE Then sweepComplete =
True
 End Sub
```

Intro to Examples

Getting Trace Data from the Analyzer

This Visual Basic program:

- Retrieves Scalar Data from the Analyzer and plots it.
- Retrieves Paired Data from the Analyzer and plots it.
- Retrieves Complex Data from the Analyzer and plots it.

To use this code, prepare a form with the following:

- Two MSCharts named **MSChart1** and **MSChart2**
- Three buttons named **GetScalar, GetPaired, GetComplex**

Note: You can get MSChart in Visual Basic by clicking **Project / Components / Microsoft Chart Control**

```
'Put this in a module
 Public dlocation As NADataStore
 Public numpts As Long
 Public fmt As NADataFormat
 Public app As Application
 Public measData As IArrayTransfer
 Public chan As Channel
Sub Form_Load()
 'Change analyzerName to your analyzer's full computer name
 Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")


Set measData = app.ActiveMeasurement
 Set chan = app.ActiveChannel

 'To pick a location to get the data from remove the comment from one of
these
 dlocation = naRawData
 'dlocation = naCorrectedData
 'dlocation = naMeasResult
 'dlocation = naRawMemory
 'dlocation = naMemoryResult
```

```
 'setup MSChart1 and MSChart2
 'right click on the chart and select:
 ' - line chart
 ' - series in rows
 End Sub


Sub GetComplex_Click()
 ReDim Data(numpts) As NAComplex
 Dim Real(201) AS Single
 Dim Imag(201) AS Single
 numpts = chan.NumberOfPoints
'You cannot change the format of Complex Data
 Call trigger
 'get data
 measData.GetNAComplex dlocation, numpts, Data(0)
 'plot data
 Dim i As Integer

 For i = 0 To numpts - 1
  Real(i) = Data(i).Re
  Imag(i) = Data(i).Im
 Next i
 MSChart1 = Real()
 MSChart2.Visible = True
 MSChart2 = Imag()
 Call Sweep
 End Sub


Sub GetPaired_Click()
 ReDim Real(numpts) As Single
 ReDim Imag(numpts) As Single
 numpts = chan.NumberOfPoints
' To pick a format, remove the comment from one of these
 fmt = naLogMagPhase
 'fmt = naLinMagPhase
 Call trigger
 'Get data
 measData.getPairedData dlocation, fmt, numpts, Real(0), Imag(0)
 'Plot Scalar
 MSChart1 = Real()
 MSChart2.Visible = True
 MSChart2 = Imag()
 Call Sweep
 End Sub


Sub GetScalar_Click()
 ReDim Data(numpts) As Single
numpts = chan.NumberOfPoints
 'To pick a format remove the comment from one of these
 fmt = naDataFormat_LogMag
 'fmt = naDataFormat_LinMag
 'fmt = naDataFormat_Phase
 'fmt = naDataFormat_Delay
 'fmt = naDataFormat_Real
```

```
 'fmt = naDataFormat_Imaginary
 Call trigger
 'Get data
 measData.GetScalar dlocation, fmt, numpts, Data(0)
 'Plot Data
 MSChart1 = Data()
 MSChart2.Visible = False
 Call Sweep
 End Sub


 Sub trigger()
'The analyzer sends continuous trigger signals
 app.TriggerSignal = naTriggerInternal
 'The channel will only accept one, then go into hold
 'Sync true will wait for the sweep to complete
sync=True
chan.Single sync
 End Sub


 Sub Sweep()
 'The channel goes back to accepting all triggers
 chan.Continuous
 End Sub
```


Limit Line Testing with COM

This Visual Basic program:

- Turns off existing Limit Lines
- Establishes Limit Lines with the following settings:
    - Frequency range - 4 GHz to 8 GHz
    - Maximum value - (10dB)
    - Minimum value - (-30dB)
- Turns on Lines, Testing, and Sound

To use this code, prepare a form with the following:

- None

```
Public limts As LimitTest
 Set limts = meas.LimitTest
 'All Off
 For i = 1 To 20
  limts(i).Type = naLimitSegmentType_OFF
 Next i

 'Set up Limit Lines
 limts(1).Type = naLimitSegmentType_Maximum
 limts(1).BeginResponse = 10
 limts(1).EndResponse = 10
 limts(1).BeginStimulus = 4000000000#
 limts(1).EndStimulus = 8000000000#
 limts(2).Type = naLimitSegmentType_Minimum
 limts(2).BeginResponse = -30
 limts(2).EndResponse = -30
```

```
limts(2).BeginStimulus = 4000000000#
limts(2).EndStimulus = 8000000000#

'Turn on Lines, Testing, and Sound
limts.LineDisplay = 1
limts.State = 1
limts.SoundOnFail = 1
```


Upload Segment Table

This Visual Basic program:

- creates a 2-dimensional array of Doubles of 7 x numSegs+1for the segment table data
- uploads the data to the PNA

The comments indicate the order in which the segment elements are specified: Index 0 - segment state, Index 4 is IFBW, and so forth.

```
Dim app As AgilentPNA835x.Application
 Dim chan As AgilentPNA835x.Channel
 Private Sub SegmentTest4_Click()
 Dim segs As AgilentPNA835x.Segments

 Const numSegs = 5 - 1 ' 5 segments
 Set chan = app.ActiveChannel ' Assumes that app is already set...
 Set segs = chan.Segments
 Dim segdata(6, numSegs)

 ' Fill up the segments with appropriate values.
 For i = 0 To numSegs
 segdata(0, i) = True ' segment state (active or not)
 segdata(1, i) = 500 'Num Points
 segdata(2, i) = 1000000# + i * 1000# 'Start Freq
 segdata(3, i) = 1000000# + i * 1000# + 300# 'Stop Freq
 segdata(4, i) = 35000# ' IFBW
 segdata(5, i) = 0# ' Dwell Time
 segdata(6, i) = 0# ' Power
 Next i

 ' Push the segment data into the PNA
 segs.SetAllSegments segdata

 End Sub
```


Microsoft Excel Example

Application Configuration

Microsoft Office 2000 was used for this example. This version of Office contains Visual Basic for Applications (VBA) which allows developers to attach Visual Basic Macros to Excel documents. The type library for the PNA network analyzer should be referenced in the Visual Basic development environment.

Application Code

The application code is contained below. The program inserts the data retrieved from the analyzer into cells in the Excel document. The cells are then used to update a graph in the Excel document. To run the application, open the document using Microsoft Excel. Enable macros when prompted by the application. Once this is complete, the application will execute and update the document. It can be run on a PC or the PNA analyzer.

```
Option Explicit

Dim app
 Dim chan
 Dim meas
 Dim result As Variant
 Dim i As Integer
 Dim num_points As Integer

Private Sub Workbook_Open()

' Connect to the PNA application; change analyzerName to your analyzer's
full computer name

Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")

' Reset the analyzer to instrument preset
 app.Reset

' Create S11 measurement
 app.CreateMeasurement 1, "S11", 1

' Set chan variable to point to the active channel
 Set chan = app.ActiveChannel

' Set meas variable to point to the active measurement
 Set meas = app.ActiveMeasurement

' Setup the channel for a single trigger
 chan.Hold True
 app.TriggerSignal = naTriggerManual
 chan.TriggerMode = naTriggerModeMeasurement

' Make the PNA application visible
 app.Visible = True

' Set channel parameters
 chan.NumberOfPoints = 11
 chan.StartFrequency = (1000000000#)
 chan.StopFrequency = (2000000000#)

' Send a manual trigger to initiate a single sweep
 chan.Single True

' Store the data in the "result" variable
 result = meas.GetData(naRawData,
 naDataFormat_LogMag)

' Display the result
 num_points = chan.NumberOfPoints 14

For i = 0 To num_points - 1
 Sheet1.Cells(3 + i, 1) = result(i)
 Next

Set chan = Nothing
 app.Quit

End Sub
```


Microsoft Visual Basic Example

Application Configuration

The type library for the PNA should be referenced in the Visual Basic development environment.

Using the Visual Basic Object Browser the developer can see what classes and methods are available for development of applications for the analyzer.

Application Code

The application code is contained below. To run the application, first generate the executable file. Once this is complete, it can be copied and executed on the analyzer or run on the PC. The application can also be run from the development environment.

```
Option Explicit

Dim app As AgilentPNA835x.Application
 Dim chan As AgilentPNA835x.Channel
 Dim meas As AgilentPNA835x.Measurement
 Dim result As Variant
 Dim i As Integer
 Dim num_points As Integer
 Dim message As String

Private Sub Main()

' Connect to the PNA application; change analyzerName to your analyzer's
full computer name

Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")

' Reset the analyzer to instrument preset
 app.Reset

' Create S11 measurement
 app.CreateMeasurement 1, "S11", 1

' Set chan variable to point to the active channel
 Set chan = app.ActiveChannel

' Set meas variable to point to the active measurement
 Set meas = app.ActiveMeasurement

' Setup the channel for a single trigger
 chan.Hold True
 app.TriggerSignal = naTriggerManual
 chan.TriggerMode = naTriggerModeMeasurement

' Make the PNA application visible
 app.Visible = True

' Set channel parameters
 chan.NumberOfPoints = 11
 chan.StartFrequency = (1000000000#)
 chan.StopFrequency = (2000000000#)

' Send a manual trigger to initiate a single sweep
 chan.Single True

' Store the data in the "result" variable
 result = meas.GetData(naRawData,
 naDataFormat_LogMag)

' Display the result
 num_points = chan.NumberOfPoints
 For i = 0 To num_points - 1
 message = message & result(i) & vbCrLf
 Next

If MsgBox(message, vbOKOnly, "S11(dB) - VBS COM
 Example for PNA") Then
 Set chan = Nothing
 app.Quit
```

```
 End If
End Sub
```

Microsoft Visual Basic Script Example

Application Configuration

Some operating systems may require that the Visual Basic Scripting engine be installed before running the application on a PC. To download a free copy of a Visual Basic Scripting engine, visit the following web site: http://msdn.microsoft.com/scripting/

Application Code

The application code is contained below. To run the program, copy the text into a text editor such as notepad and save the file with the ".vbs" extension. The ".vbs" extension will tell the operating system to execute the code using the Visual Basic Scripting engine.

In order to run the application, double-click on the saved .vbs file. The application can be run on a PC or on the PNA Series network analyzer.

```
Option Explicit

' Shell objects
 Dim app
 Dim chan
 Dim meas
 Dim result
 Dim message
 Dim num_points
 Dim i

' Connect to the PNA application; change analyzerName to your analyzer's
full computer name

Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")

' Reset the analyzer to instrument preset
 app.Reset

' Create S11 measurement
 app.CreateMeasurement 1, "S11", 1

' Set chan variable to point to the active channel
 Set chan = app.ActiveChannel

' Set meas variable to point to the active measurement
 Set meas = app.ActiveMeasurement

' Setup the channel for a single trigger
 chan.Hold True
 app.TriggerSignal = 3
 chan.TriggerMode = 1

' Make the PNA application visible
 app.Visible = True
 ' Set channel parameters
 chan.NumberOfPoints = 11
 chan.StartFrequency = (1000000000)
 chan.StopFrequency = (2000000000)

' Send a manual trigger to initiate a single sweep
 chan.Single True
```

```
' Store the data in the "result" variable
 result = meas.GetData(0, 1)

' Display the result
 num_points = chan.NumberOfPoints
 For i = 0 To num_points - 1
 message = message & result(i) & vbCRLF
 Next

if MsgBox(message, vbOKOnly, "S11(dB) - VBS COM
 Example for PNA") then
 Set chan = Nothing
 app.quit

end if
```


Microsoft Visual C++ Example

Application Configuration

Microsoft Visual C++ version 6 was used for this example. In order to perform this example, create a new project in Microsoft Visual C++. Add a C++ file to the project and paste the following code into the file. The path for the type library in the code below should be changed to reference its location on the development PC.

Application Code

The application can be run on a PC or on the PNA.

```
#include "stdafx.h"
```

```
// import the Tsunami type library
 //--------------------------
 #import "C:\Program Files\Common Files\Agilent\Pna\835x.tlb"
 no_namespace, named_guids
 int main(int argc, char* argv[])
 {
 // interface pointers to retrieve COM interfaces
 IUnknown* pUnk = 0;
 IApplication* pNA = 0;
 IChannel* pChan = 0;
 IMeasurement* pMeas = 0;
 IArrayTransfer* pTrans = 0;
 int i, num_points = 0;
 float* pScalarData;

HRESULT hr;

// Initialize the COM subsystem
 CoInitialize(NULL);

// Create an instance of the network analyzer
 // Request the NA's IUnknown interface
 hr = CoCreateInstance(CLSID_Application,0,
 CLSCTX_ALL,IID_IUnknown, (void**) &pUnk);
 if (!FAILED(hr)) {

// QueryInterface for the INetworkAnalyzer interface
 of the NetworkAnalyzer object
```

```
 hr = pUnk->QueryInterface(IID_IApplication,
 (void**)&pNA);
if (!FAILED(hr)) {
// Reset the analyzer to instrument preset
 pNA->Reset();
// Create S11 measurement
 pNA->CreateSParameter(1,1,1,1);
// Set pChan variable to point to the active
 channel
 pNA->get_ActiveChannel(&pChan);
if (pChan) {
// Set pMeas variable to point to the active
 measurement
 pNA->get_ActiveMeasurement(&pMeas);
if(pMeas) {
// Setup the channel for a single trigger
 pChan->Hold(true);
 pNA->TriggerSignal = naTriggerManual;
 pChan->TriggerMode =
 naTriggerModeMeasurement;
// Make the PNA application visible
 pNA->put_Visible(true);
// Set channel parameters
 pChan->NumberOfPoints = 11;
 pChan->StartFrequency = 1e9;
 pChan->StopFrequency = 2e9;
// Send a manual trigger to initiate a single
 sweep
 pChan->Single(true);
// QueryInterface for the IArrayTransfer
 interface of the NetworkAnalyzer object
 hr = pMeas->QueryInterface(IID_IArray
 Transfer, (void**)&pTrans);
if (!FAILED(hr)) {
// Store the data in the "result" variable
 num_points = pChan->NumberOfPoints;
 pScalarData = new float[num_points];
 pTrans->getScalar(naRawData, naData
 Format_LogMag, (long *)&num_points,
 pScalarData);
// Display the result
 printf("S11(dB) - Visual C++ COM
 Example for PNA\n\n");
 for (i = 0; i < num_points; i++)
 printf("%f\n",pScalarData[i]);
}
 }
 }
 }
pUnk->Release();
 pMeas->Release();
 pChan->Release();
 pTrans->Release();
 pNA->Release();
 }
 CoUninitialize();
```

```
 return 0;
 }
```

____

Microsoft Word Example

____

Application Configuration

Microsoft® Office 2000 was used for this example. This version of Office contains Visual Basic for Applications (VBA) which allows developers to attach Visual Basic Macros to Word documents. The type library for the PNA Series network analyzer should be referenced in the Visual Basic development environment.

Application Code

The application code is contained below. The program inserts the data retrieved from the analyzer into a table in a Word document. To run the application, open the document using Microsoft Word. Enable the macros when prompted. Once this is complete, the application will execute and update the document. The application can be run on a PC or the analyzer.

____

```
Option Explicit

Dim app
 Dim chan
 Dim meas
 Dim result As Variant
 Dim i As Integer
 Dim num_points As Integer

Private Sub Document_Open()

' Connect to the PNA application; change analyzerName to your analyzer's
full computer name

Set app = CreateObject("AgilentPNA835x.Application", "analyzerName")

' Reset the analyzer to instrument preset
 app.Reset

' Create S11 measurement
 app.CreateMeasurement 1, "S11", 1

' Set chan variable to point to the active channel
 Set chan = app.ActiveChannel

' Set meas variable to point to the active measurement
 Set meas = app.ActiveMeasurement

' Setup the channel for a single trigger
 chan.Hold True
 app.TriggerSignal = naTriggerManual
 chan.TriggerMode = naTriggerModeMeasurement

' Make the PNA application visible
 app.Visible = True

' Set channel parameters
 chan.NumberOfPoints = 11
 chan.StartFrequency = (1000000000#)
 chan.StopFrequency = (2000000000#)

' Send a manual trigger to initiate a single sweep
 chan.Single True

' Store the data in the "result" variable
```

```
 result = meas.GetData(naRawData,
 naDataFormat_LogMag)
' Display the result
 num_points = chan.NumberOfPoints
 For i = 0 To num_points - 1
 ThisDocument.Tables(1).Cell(i + 2, 2).Range = result(i)
 Next
Set chan = Nothing
 app.Quit
End Sub
```


National Instruments™ LabVIEW Example

Application Configuration

Use National Instruments™ Lab VIEW version 5.0 or above for this example. See the National Instruments™ LabVIEW documentation for information on using ActiveX objects in the LabVIEW development environment.

Application Code

National Instruments™ LabVIEW 5.0 or higher must be installed to run the application. The application can be run on a PC or on the PNA Series analyzer.

The application file is located at http://agilent.com/find/pna_applications.

Learning about COM

Learning about COM

The following topics can help you learn more about controlling the PNA using COM.

- COM versus SCPI
- Configure for COM-DCOM Programming
- COM Fundamentals
- Getting a Handle to an Object
- COM Collections in the PNA
- COM Data Types
- Working with PNA Events
- COM for Multiple PNA Versions
- Read and Write Calibration Data using COM
- C++ and the COM Interface
- Using COM with .NET

Configure for COM-DCOM Programming

Before developing or running a COM program, you should first establish communication between your PC and the analyzer. This process is referred to as gaining **Access** to the analyzer. Then, to work with the analyzer's components, you should register the PNA type library on your PC.

- Access Concepts
- Access Procedures
- Register the PNA Type Library on Your PC
- Problems?
- Using the FCA Type Library

Note: After upgrading the Network Analyzer application, you must also copy the new type library to your development PC to get access to new COM commands. See Register the analyzer on your PC.

Other Topics about COM Concepts

Access Concepts

PNAs are shipped from the factory such that **Everyone** has permission to launch and access the PNA application via COM/DCOM. The term **Everyone** refers to a different range of users depending on whether the PNA is a member of a **Domain** or **Workgroup** (it must be one or the other; not both). By default, the PNA is configured as members of a workgroup. Therefore, **Everyone** includes only those users who have been given logon accounts on the PNA.

Note: DCOM (Distributed Component Object Model) refers to accessing the analyzer application from a remote PC. **COM** refers to accessing the analyzer application from the analyzer PC.

Workgroup

A workgroup is established by the **PNA administrator** declaring the workgroup name and declaring the PNA as a member of the workgroup. A workgroup does not require a network administrator to create it or control membership.

Everyone includes only those users who have been given logon accounts on the PNA.

By default, the PNA is configured as members of a workgroup named WORKGROUP.

Note: To setup a logon account for a new user, see Additional Users.
For DCOM access, the user's account name and password must EXACTLY match their PC logon account name and password.

Domain

A domain is typically a large organizational group of computers. Network administrators maintain the domain and control which machines have membership in it.

Everyone includes those people who have membership in the domain. In addition, those with logon accounts can also access the analyzer.

Summary

- A **Workgroup** requires no maintenance, but allows DCOM access to only those users with a log-on account for the PNA.
- A **Domain** requires an administrator, but all members of the domain and those with logons to the analyzer are allowed DCOM access to the PNA.

The next level of security is to allow only **selected** (not **Everyone**) domain and workgroup users DCOM **Access** and **Launch** capability of the analyzer.

Access Procedures

Perform this procedure for the following:

- To allow only selected users (not everyone) remote Access and Launch capability to the analyzer. Launch capability is starting the analyzer application if it is not already open.
- To verify that you have DCOM access to the analyzer.

Note: Before doing this procedure, you must first have a logon account on the PNA. See Additional Users

---

Do the following for both Access and Launch capabilities:

1. On the PNA, click the Windows **Start** button
2. Click **Run**
3. In the **Open:** box, type **dcomcnfg**
4. Click **OK**
5. In the Distributed COM Configuration Properties window, Click on **Agilent PNA Series** in the Applications list. Then click **Properties...** (button)
6. Click the **Security** tab

Access Capability

The following configures the PNA to grant specific users **DCOM access** to the PNA application:

in the **Agilent PNA Series Properties** dialog box:

1. Click **Use custom access permissions**
2. Click **Edit** next to **(Use custom access permissions)**
3. In Registry Value Permissions, select **Everyone**
4. Click **Remove**
5. Click **Add**
6. You could either select one or more of these groups to have access to the PNA, or specific users.
7. To give groups access, select the group from the list.
8. To give specific users access, click **Show users** or **Members**, then select the name from the list.
9. Click **Add**
10. Click **OK**

Launch Permission

The following configures the PNA to allow selected users to **Launch** (start) the PNA application.

In **Agilent PNA Series Properties:**

1. Click **Use custom launch permissions**
2. Click **Edit** (next to **Use custom launch permissions)**
3. In Registry Value Permissions, select **Everyone**
4. Click **Remove**
5. Click **Add**
6. You could either select one or more of these groups to have launch permission of the PNA, or specific users.
    - To give groups launch permission, select the group from the list.
    - To give specific users launch permission, click **Show users** or **Members**, then select the

name from the list.

7. Click **Add**
8. Click **OK**

In **Agilent PNA Series Properties:**

1. Click the **Identity tab.**
2. Click **The interactive user**. This function supports Events in PNA COM.

Register the PNA Type Library on Your PC

The type library contains the PNA object model. On your PC, there is a Registry file that keeps track of where object models are located. Therefore, you must register the type library on the PC that will be used to develop code and run the program. It is much more efficient to have the type library registered at design time (BEFORE running your COM program).

Do the following two items before proceeding:

1. Connect your PC and the PNA to LAN.
2. Either map a drive to the analyzer or copy the type library files on a floppy disk or other media. See Drive Mapping.

Note: To register the type library on your PC, you must be logged on as an administrator of your PC.

This procedure will do the following:

- Register the Network Analyzer application on your PC.
- Copy and register the proxystub (835xps.DLL) onto the PC.
- Copy and register the type library (835x.tlb) onto the PC.

1. Using Windows Explorer on your PC, find the Analyzer's C: drive. The drive will not be named "C:" on your PC, but a letter you assigned when mapping the drive.
2. Navigate to **Program Files \ Agilent** \ **Network Analyzer \ Automation**
3. Double-click **pnaproxy.exe**
4. The install program will ask for the full computer name of your PNA. (You can find this at **Control Panel, System, Network Identification, Full Computer name**.) Type the Analyzer name at the prompt.

Note: The process will fail if the type library is currently being used by a development environment on the PC.

5. After the install program runs, the analyzer type library should be registered on your PC.

Note: Your programming environment may require you to set a reference to the PNA type library now located on your PC. In Visual Basic, click **Project, References**. Then browse to **C:\Program Files\Common Files\Agilent\PNA** Select **835x.tlb**

Problems?

Perform the following procedure if the previous procedure did not return an error, but you cannot connect to the PNA.

If you received an error, check that both the account name and password used on both the PNA and PC match EXACTLY.

Note: The previous procedure and the following procedure will both fail if there are any programs using the PNA type library. For example: Visual basic, VEE, Visual Studio, or any other application program that may communicate with the PNA.

1. Map a drive from your remote PC to the PNA. Note the drive letter your PC assigns to the PNA. Substitute this drive letter for **PNA** in the following procedure.
2. On your PC, go to a DOS prompt c:>
3. Type **cd PNA:\program files\agilent\network analyzer**
4. Type **copy 835xps.dll c:\program files\common files\agilent\pna**
5. Type **cd automation**
6. Type **copy 835x.tlb c:\program files\common files\agilent\pna**
7. If it is not already there, copy **regtlib.exe** from **PNA**:\WINNT to your C:\<windows>\system32 directory
8. (<windows> is OS-dependent- it is either windows or WINNT)
9. Type **regtlib "C:\program files\common files\agilent\pna\835x.tlb"**
10. Type r**egsvr32 "C:\program files\common files\agilent\pna\835xps.dll"**

After doing these, perform "Access Procedure" (run dcomcnfg).

Using the FCA Type Library

You can control the Frequency Converter Application (opt 083) using COM. This application has its own type library; it is not included with the PNA type library. To use the type library on your remote PC for development, copy the type library from the following location on the PNA to your remote PC:
C:\Program Files\Agilent\Network Analyzer\Applications\fca\fcapplib.tlb

You can then reference the FCA type library from your programming environment.

COM Fundamentals

The following terms are discussed in this topic:

- Objects
- Collections
- Methods
- Properties
- Events

Note: The information contained in this topic is intended to help an experienced SCPI programmer transition to COM programming. This is NOT a comprehensive tutorial on COM programming.

Other Topics about COM Concepts

Objects

The objects of the Network Analyzer (Application) are arranged in a hierarchical order. The Network Analyzer object model lists the objects and their relationship to one another.

In SCPI programming, you must first select a measurement before making settings. With COM, you first get a handle to the object (or collection) and refer to that object in order to change or read settings.

For more information on working with objects, see Getting a Handle to an Object.

Collections

A collection is an object that contains several other objects of the same type. For example, the **Channels** collection contains all of the channel objects.

Note: In the following examples, the collections are referred to as a variable. Before using a collection object, you must first get an instance of that object. For more information, see Getting a Handle to an Object

---

Generally, items in a collection can be identified by **number** or by **name**. The order for objects in a collection cannot be assumed. They are always unordered and begin with 1. For example, in the following procedure, chans(1) is used to set averaging on the **first** channel in the Channels collection (not necessarily channel 1).

```
Sub SetAveraging()
   chans(1).AveragingFactor = 10
 End Sub
```

The following procedure uses the measurement string name to set the display format for a measurement in the measurements collection.

```
meass("CH1_S11_1").Format = 1
```

You can also manipulate an entire collection of objects if the objects share common methods. For example, the following procedure sets the dwell time on all of the segments in the collection.

```
Sub setDwell()
   segs.DwellTime = 30e-3
 End Sub
```

Methods

A method is an action that is performed on an object. For example, **Add** is a method that applies to the Channel object. The following procedure uses the Add method to add a new channel named **NewChan.**

```
Sub AddChan(newChan as String)
   Chan.Add NewChan
 End Sub
```

Properties

A property is an attribute of an object that defines one of the object's characteristics, such as size, color, or screen location. A property can also change an aspect of the object's behavior, such as whether the object is visible. In either case, to change the characteristics of an object, you change the values of its properties.

To change the value of a property, follow the reference to an object with:

- a period (.)
- the property name
- an equal sign (=)
- the new property value.

For example, the following statement sets the IFBandwidth of a channel.

```
Chan.IFBandwidth = 1KHz
```

You can also read the current value of a property. The following statement reads the current IFBandwidth of a channel into the variable **Ifbw.**

```
Ifbw = Chan.IFBandwidth
```

Some properties cannot be set and some cannot be read. The Help topic for each property indicates if you can:

- Set and read the property (Write/Read)
- Only read the property (Read-only)
- Only set the property (Write-only)

Events

An event is an action recognized by an object, such as clicking the mouse or pressing a key. Using events, your program can respond to a user action, program code, or triggered by the analyzer. For example:

```
OnChannelEvent
```

For more information, see Working with the Analyzer's Events.

Collections in the Analyzer

Collections are a gathering of similar objects. They are a convenience item used primarily to iterate through the like objects in order to change their settings. Collections generally provide the following generic methods and properties:

```
Item(n)
 Count
 Add(n)
 Remove(n)
```

where **(n)** represents the number of the item in the collection. Some collections may have unique capabilities pertinent to the objects they collect.

Other Topics about COM Concepts

Collections are Dynamic

A collection does not exist until you ask for it. When you request a Channels object (see Getting a Handle to an Object / Collection), handles to each of the channel objects are gathered and placed in an array.

For example, if channels 2 and 4 are the only channels that exist, then the array will contain only 2 items. The command 'channels.Count' will return the number 2, and:

- Channels(1) will contain the channel 2 object.
- Channels(2) will contain the channel 4 object.

The ordering of objects within the collection should not be assumed. If you add a channel to the previous example, as in:

```
Pna.Channels.Add(3)
```

'channels.Count' will now return 3 and:

- Channels(1) will contain the channel 2 object.
- Channels(2) will contain the channel 3 object.

- Channels(3) will contain the channel 4 object.

Primarily, collections are useful for making this type of iteration possible:

```
Dim ch as Channel
 For each ch in pna.Channels
   Print ch.Number
   Print ch.StartFrequency
   Print ch.StopFrequency
 Next ch
```

As soon as this for-each block has been executed, the Channels object goes out of scope.

COM Data Types

The PNA uses several data types to communicate with the host computer. Before using a variable, it is best to declare the variable as the type of data it will store. It saves memory and is usually faster to access. The following are the most common data types:

- Long Integer
- Single Precision (Real)
- Double Precision (Real)
- Boolean
- String
- Object
- Enumeration
- Variant

Other Topics about COM Concepts

Long (long integer) variables are stored as signed 32-bit (4-byte) numbers ranging in value from -2,147,483,648 to 2,147,483,647.

Double (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers ranging in value from -1.79769313486232E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values.

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values.

Boolean variables are stored as 16-bit (2-byte) numbers, but they can only be True or False. Use the keywords True and False to assign one of the two states to Boolean variables.

When other numeric types are converted to Boolean values, 0 becomes False and all other values become True. When Boolean values are converted to other data types, False becomes 0 and True becomes -1.

String variables hold character information. A String variable can contain approximately 65,535 bytes (64K), is either fixed-length or variable-length, and contains one character per byte. Fixed-length strings are declared to be a specific length. Variable-length strings can be any length up to 64K, less a small amount of storage overhead.

Object variables are stored as 32-bit (4-byte) addresses that refer to objects within the analyzer or within some other application. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual analyzer object.

Enumerations (Enum) are a set of named constant values. They allow the programmer to refer to a constant value by name instead of by number. For example:

```
Enum DaysOfWeek
   Sunday = 0
   Monday = 1
   Tuesday = 2
   Wednesday = 3
   Thursday = 4
   Friday = 5
   Saturday = 6
 End Enum
```

Given this set of enumerations, the programmer can then pass a constant value as follows:

```
SetTheDay(Monday)
```

rather than

```
SetTheDay(1)
```

where the reader of the code has no idea what the value 1 refers to.

However, the analyzer RETURNS a long integer, not the text.

```
Day = DaysofWeek(today) 'Day = 1
```

Variant - If you don't declare a data type ("typed" data) the variable is given the Variant data type. The Variant data type is like a chameleon — it can represent many different data types in different situations.

The PNA provides and receives Variant data because there a programming languages that cannot send or receive "typed" data. Variant data transfers at a slower rate than "typed" data.

Getting a Handle to an Object

The following are discussed in this topic:

- What Is a Handle
- Declaring an Object Variable
- Assigning an Object Variable
- Navigating the Object Hierarchy
- Getting a Handle to a Collection

Other Topics about COM Concepts

What Is a Handle

In SCPI programming, you must first select a measurement before changing or reading settings. With COM, you first get a handle to the object (or collection) and refer to that object to change or read its settings. The following analogy illustrates this:

A car could be called an object. Like all objects, it has many properties. One of its **properties** is "**Color**". You can read (by looking) or set (by painting) the color property of a car object. However, the color **value** (such as **Red** or **Green**) depends on what SPECIFIC car object you are referring to. "Car" is actually a **class** of objects. You can only read or set the properties of a specific car object; not the entire car class. Therefore, before reading or setting an object's properties, you need to get "a handle" to a specific object.

You can have handles to many objects at the same time. It does NOT have to be the Active or Selected object.

Note: This process is also called "getting an instance of an object", "returning an object". or "referring to an object"

There are two steps for getting a handle to analyzer objects:

1. Declaring a Variable As an Object
2. Assigning an Object to the Variable

Note: Before doing this, you must first register the analyzer's type library on your PC. See Connecting to the Analyzer

Declaring a Variable As an Object

Note: The examples in these topics use the Visual Basic Programming Language. The **Green** text following an apostrophe (') is a comment.

Use the Dim statement or one of the other declaration statements (Public, Private, or Static) to declare a variable. The type of variable that refers to an object must be a Variant, an Object, or a specific type of object. For example, all three of the following declarations are valid:

- **`Dim RFNA ' Declare RFNA as Variant data type.`**
- Dim RFNA As Object ' Declare RFNA as Object data type.
- Dim RFNA As AgilentPNA835x.Application ' Declare RFNA As AgilentPNA835x.Application type

Note: If you use a variable without declaring it first, the data type of the variable is Variant by default.

If you know the specific object type, you should declare the object variable as that object type. Declaring specific object types provides automatic type checking, faster code, and improved readability.

Assigning an Object to a Variable

The first and most important object to assign to a variable is the Application object (the Network Analyzer). When assigning an object to a variable, use the **Set** keyword before the object variable that was declared previously. In the following example, "RFNA" is the variable we declared in the previous examples. So we assign the current AgilentPNA835x Application to "RFNA".

```
Set RFNA = AgilentPNA835x.Application
```

However, because the AgilentPNA835x object is the Application server, we must use the **CreateObject** keyword with the (*classname,server name*) parameters.

- The ***classname*** for the analyzer object is always "AgilentPNA835x.Application".
- To find your analyzer's ***server name***, see Sharing Files between your PC and the Analyzer.

For example, the following statements would create an instance of the Analyzer object.

```
Dim RFNA AS AgilentPNA835x.Application
 Set RFNA = CreateObject("AgilentPNA835x.Application", "Analyzer46")
```

Note: These statements will start the Analyzer application if it is not already running on your instrument.

Once created, you can treat an object variable exactly the same as the object to which it refers. You can set or return the properties of the object or use any of its methods. For example:

```
RFNA.Visible = True 'Makes the Network Analyzer Application visible on
the screen
```

Navigating the Object Hierarchy

To read and set properties of objects below the Analyzer Application, you do not have to "Create" the object as we did with the Application. But you DO have to navigate the object model hierarchy. (Refer to the Analyzer Object Model).

You could do refer to an object in the hierarchy directly, without declaring and assigning a variables. The following example navigates through the Application object to the Active Measurement which is a 'child' object of the Application. (The ACTIVE measurement is the measurement that is acted on if you change settings from the front panel.)

```
Application.ActiveMeasurement.SmoothingAperture = 10
```

You can see that this method makes for a very long statement. Making additional changes to the Active Measurement would require equally long statements.

The following example gets a handle to the Active Measurement object by assigning it to a variable.

The first step is to **Declare an object variable:**

```
Public meas AS Measurement
```

The next step is to **Set the object variable:**

We already assigned an instance of the (analyzer) Application to the variable **RFNA**. Therefore, we can use the RFNA variable to refer to a specific instance of the Application object.

```
Set meas = RFNA.ActiveMeasurement
```

The variable **meas** now contains a handle to the Application object (RFNA) **and** the ActiveMeasurement object. We can now set properties of the ActiveMeasurement as follows:

```
meas.SmoothingAperature = 10
```

Getting a Handle to a Collection

The analyzer has several collections of objects which provide a convenient way of setting or reading all of the objects in the collection with a single procedure. Also, there are objects (limit lines for example) that can only be accessed through the collection.

To get a handle to an item in a collection, you can refer to the object by item number or sometimes by name. However, you first have to get a handle to the collection. To assign the collection to a variable, use the same two step process (1. declare the variable, 2. assign the variable using 'Set').

```
Dim meass As Measurements 'the collection of all measurements currently
on the analyzer
 Set meass = RFNA.Measurements
```

Then you can iterate through the entire collection of measurements to read or set properties or execute methods.

```
meass.Format = naLinMag
```

Or you can read or set a property on an individual object in the collection:

```
meass(1).Format = naLinMag
```

Note: Each object and collection has its own unique way of dealing with item names, and numbers. Refer to the Analyzer Object Model for details.

Programming the PNA with C++

The programming information contained in this Help system is aimed at the Visual Basic programmer. VB does a lot of work for the programmer when it comes to managing and accessing components. Using a lower level language like C++ requires a more thorough understanding of the underlying tenets of COM. It is not the intent of this section to teach COM programming. The following is intended to acquaint you with some of the basic concepts you need to know in order to program against COM.

- Initializing COM
- Importing the Type Library
- Creating the Application Object
- Errors
- Events
- Additional Reading
- Example

Note: The information in this section assumes development on a Windows OS using Microsoft tools.

Other Topics about COM Concepts

Initializing COM

The first thing you must do before performing any COM transactions is to initialize the COM library. You can do this in a number of ways. The most basic of these is a call to **CoInitialize( )** or **CoIntializeEx( )**. Alternatively you can use the MFC (Microsoft Foundation Classes) **AfxOleInit( )**.

Conversely, before your program exits you must uninitialize COM. You can accomplish this with **CoUninitialize( )** or the MFC routine **AfxOleTerm( )**.

Importing the Type Library

To make a component available to the client, the server exports what is called the type library. For the PNA, this file is 835x.tlb. It is located on the PNA's hard drive at **C:\Program Files\ Agilent\ Network Analyzer\ Automation**. See Configure for COM-DCOM Programming.

The type library can be read and deciphered using another COM interface called ITypeLib. VB uses this interface to present, for example, its object browser. Visual C++ can also read type libraries. This is done by importing the type library into your project with a compiler directive:

```
#import "C:\Program Files\Common Files\Agilent\Pna\835x.tlb",
named_guids
```

When you compile your program with this statement in it, the compiler creates two other files: **835x.tlh** and **835x.tli**. The first is a header file that contains the type definitions for the PNA's COM interfaces and their methods. The second file contains inline functions that wrap the PNA's interface methods. The wrappers are beneficial in that they contain error reporting for each of the method calls.

The .tlh file defines a smart pointer which you can use to access the PNA's objects. The smart pointer definition looks like this:

```
 _com_smartptr_typedef(Iapplication, _uuidof(Iapplication))
```

A smart pointer is a term used for a C++ object that encapsulates a pointer used to refer to a COM object. All COM objects derive from the interface IUnknown. This interface has three methods: QueryInterface( ), AddRef( ), and Release( ). The function of the AddRef and Release methods is to maintain a reference count on the object and thus control the object's lifetime. Anytime you copy or create a reference to a COM object, you are responsible for incrementing its reference count. And likewise, when you are finished using that reference, it is your responsibility to Release it. Smart pointers do this work for you, as shown in the example program. In addition, smart pointers will also perform the QueryInterface call when required. QueryInterface is a method that requests a specific interface from an object. In the example program we gain access to the IArrayTransfer interface of the Measurement object. In the ReadMethod routine, we see this:

```
PTransferData = pMeas;
```

The assignment operator is overloaded for the smart pointer and in reality, this simple statement does this:

```
HRESULT hr = pMeas->QueryInterface(
IID_IArrayTransfer,(void**)&pTransferData);
```

Using the existing interface pointer (pMeas) to the object, this call asks the object if it supports the IArrayTransfer interface, and if so to return a pointer to it in pTransferData. Smart pointer makes life easier for the C++ programmer. Read more about smart pointers in Microsoft Developer's Network Library *(MSDN).*

Creating the Application Object

The only createable object exported by the PNA is the Application object. Typically this would be done with a call to CoCreateInstance:

```
STDAPI CoCreateInstance(
   CLSID__IApplication, //Class identifier (CLSID) of the object
   NULL, //Pointer to controlling IUnknown
   CLS_CTX_SERVER, //Context for running executable code
   IID_IApplication, //Reference to the IID of the interface
   (void**)&pNA //Address of output variable that receives
   // the interface pointer requested in riid
 );
```

With the smart pointer, this is taken care of with the following call:

```
IApplicationPtr pNA; // declare the smart pointer
 pNA = IApplicationPtr("AgilentPNA835x.Application.1");
```

Errors

All COM method calls are required to return an HRESULT. This is 32 bit long with a specific format.

- The most significant bit indicates success(0) or failure(1).
- The lower 16 bits indicate the specific failure.

Visual Basic strips off the returned HRESULT and raises an error object for non-successful returns. The C++ programmer must himself be diligent about handling errors. You must check the return value of each COM call to ensure its success.

Events

The Application object sources the INetworkAnalyzerEvents interface. This object is the source for all events. To use events in C++, you must do two things:

1. Implement the INetworkAnalyzerEvents interface - derive an object from INetworkAnalyzerEvents and implement the methods described there.
2. Subscribe to the IconnectionPoint interface of the Application object. - obtain a pointer to the IConnectionPointContainer interface of the Application object and making the following request:

```
FindConnectionPoint( IID_InetworkAnalyzerEvents, &pConnection );
```

A successful call to this interface will return a valid pointer in pConnection. Use this pointer to subscribe to the Application object:

```
pConnect->Advise( IUnknown* punk, DWORD dwCookie);
```

This call provides the server object with a callback address. The Iunkown pointer in this call is the IUnkown pointer of the object that implements the INetworkAnalyzerEvents interface. This is the event sink. The application object needs a pointer to this object in order to call your interface when an event occurs. The **dwCookie** is your subscription key. Use it to unsubscribe (see Unadvise( )).

Additional Reading

"MSDN" - Microsoft Developer's Network Library

"Learning DCOM", by Thuan L. Thai, published by O'Reilly(1999)

"Inside COM", by Dale Rogerson, published by Microsoft Press (1997)

"Understanding ActiveX and OLE", by David Chappell, also published by Microsoft Press (1996)

"Beginning ATL COM Programming", published by Wrox Press (1998)

Example

The example uses the smart pointer created by Microsoft Visual Studio. The calls to CoInitialize and CoUninitialize open and close the COM libraries. In the example, notice that the pointers local to the main routine are explicitly released. When smart pointers go out of scope, they will perform this duty implicitly. However, we are calling CoUninitialize before they have the chance to be destroyed, so we are obliged to release them.

See the example program.

Writing COM Code for Multiple PNA Versions

The PNA continues to evolve and release new firmware / software versions that provides more functionality and features. In new releases, we add COM commands to allow remote access to these new features. To ensure backward compatibility, the new commands are added to new Interfaces, which extend and inherit from the existing objects. This guarantees that the code that you wrote for older PNA versions continues to run error-free on new PNA versions.

Developing Programs to Run on Multiple PNA Versions

The remainder of this topic discusses how you can successfully develop code in an environment that requires a single program to run on multiple versions (older and newer) of the PNA.

You can NOT develop code utilizing new features and run the code on older PNA versions. For example, errors will occur if you use the **CopyChannel** method, that was introduced in PNA 3.0, on instruments that are running PNA 2.0. The PNA will not recognize that feature.

Even if you do not utilize new features, you can still have problems if developing with a newer type library than the code will be running on. The following example shows how this occurs and how to avoid it:

Visual Basic allows you to dimension variables as an object, even though it will convert the object to the default interface during compilation. For example, the following statement will refer to the default Interface of the Measurement Object:

```
Dim meas as Measurement
```

If developed using the PNA 3.0 type library, the default Interface of the Measurement Object is IMeasurement2. If this statement is run on an instrument with PNA 2.0 firmware, an E_NOINTERFACE error will occur because there was no IMeasurement2 Interface at the time of PNA 2.0.

Instead, specify the Interface:

```
Dim meas as IMeasurement
```

Although you will not be able to access the newest commands, at least it will not result in errors when run on older PNA versions.

Read and Write Calibration Data using COM

You can read or write two types of Calibration data in the PNA:

- **Standard Measurement data** -raw data resulting from the measurement of a calibration standard.
- **Error Terms** - calculated data using standard measurement data and the algorithms for the specified cal type.

Each of these data are available in the PNA in either variant data or typed data. Learn more about variant and typed data

Other Topics about COM Concepts

Evolution of the Calibration Architecture

PNA 2.0 expanded the use of the Cal Set, which is simply a container for calibration data. In PNA 1.0 the Cal Set was restricted to one cal type and could only be used by the channel that created it. In PNA 2.0, the Cal Set is sized dynamically, can accommodate more than one cal type, and can be used by multiple channels. (Learn more about Cal Sets)

The PNA has two sets of automation interfaces that contain methods for getting and putting Calibration data in a Cal Set:

Set 1 - ICalibrator (variant), ICalData (typed)

The ICalibrator and ICalData interfaces were introduced in PNA 1.0. They contain several methods for putting and getting error terms and standard measurement data.

Set 2 - ICalSet (variant), ICalData2(typed)

The ICalSet interface was introduced with PNA2.0 to support the new Cal Set features. The methods on this interface include, but are not limited to, putting and getting data to and from the

Cal Set. In addition, the ICalData2 interface was introduced to work with non-variant data. The following is an example of using ICalSet to read error term data. This examples gets a handle to a Cal Set using the GetCalSetByGUID method.

```
dim CMGR as CalManager
 dim CSet as CalSet
 dim strCalSetGUID as string
 dim iEtermSetID as integer
 dim caltype as NACalType
 dim eTerm as NAErrorTerm2
 dim rcvPort as long
 dim srcPort as long
 CMGR.GetCalSetUsageInfo( channel, strCalsetGUID, iEtermSetID)
 set CSet = CMGR.GetCalSetByGUID( strCalSetGUID)
 caltype = naResponseOpen
 rcvPort = 1
 srcPort = 1
 eTerm = naET_ReflectionTracking
 CSet.Open( caltype, rcvPort, srcPort)
 VarData = CSet.GetErrorTerm( ETerm, rcvPort, srcPort)
 CSet.Close()
```

Recommendation

For reading and writing calibration data, we strongly recommend using the ICalSet and ICalData2.

Note: The ICalibrator interface still required for other calibration activities, such as acquiring calibration data.

Using ICalibrator with PNA2.0 Cal Sets

You can still use the ICalibrator interface to read and write calibration data on the 2.0 Cal Sets.

To data from a Cal Set,

1. Get a handle to the Cal Set using one of the "get" methods on the ICalManager Interface
2. Get a handle to a Calibrator object on the same channel as the Cal Set.
3. Specify the Cal Type and ports with the SetCalInfo method:

The following example reads error term data from a Cal Set

```
Need code here that gets a handle to a Cal Set

ICalibrator.SetCalInfo( caltype, rcvPort, srcPort)
 VarData = ICalibrator.GetErrorTerm( ETerm, rcvPort, srcPort)
```

Write data to a Cal Set

You can either fill an "empty" cal set with data or overwrite an existing Cal Set. The SetCalInfo method will create an empty Cal Set if there is no Active Cal Set on the same channel as the Calibrator object. The following example writes error terms to an empty Cal Set.

```
ICalibrator.SetCalInfo(  caltype, rcvPort, srcPort)
 VarData = ICalibrator.putErrorTerm( ETerm, rcvPort, srcPort)
```

Using COM from .NET

To communicate with the PNA from Microsoft .NET enabled languages such as C# and Visual Basic.NET perform the following steps:

1. Configure your PC and PNA for COM-DCOM Programming.
2. Reference the type library within the development environment (see the following exception for managed C++ projects.) In the process of referencing the type library, a .NET assembly is created that wraps the PNA type library with a .NET friendly interface. This .NET assembly is called an Interop Assembly.

   Exception for managed C++ projects To generate the Interop Assembly for managed C++ projects, you must use the tlbimp.exe utility. This utility is described in the MSDN documentation. On your PC, click **Start** then **Run** then type: **tlbimp.exe 835x.tlb** and click **OK.** After doing this you can use the #using directive to include the Interop Assembly on managed C++ projects.

Registering the PNA Primary Interop Assembly (PIA) (OPTIONAL)

The PIA is NOT necessary to communicate with the PNA. The following procedure is useful only when there are two .NET programs that want to share the same PNA interface definitions. Without the PIA, each .NET application would use its own Interop Assembly.

To register the PIA on a machine, you need to have the common language runtime (CLR) installed. This is included with Visual Studio.NET. Then perform the following steps:

Note: In the following steps, replace <local directory> with the full path name of the specified file on your PC.

1. Run the PNAProxy.exe program as described in Configure for COM-DCOM Programming.
2. On the PNA, copy **C:\Program Files\Agilent\Network Analyzer\Automation\AgilentPNA835x.dll** to a local directory on your PC. Make a note of this directory.
3. On your PC, click **Start**, then **Run**, then type: **regasm** <local directory> **\AgilentPNA835x.dll** and click **OK** to register the dll.
4. Again, click **Start**, then **Run**, then type: **gacutil /i** <local directory> **\AgilentPNA835x.dll** and click **OK** to add the assembly to the Global Assembly Cache (GAC).

To **Uninstall the PIA**, perform the following:

1. On your PC, click **Start**, then **Run**, then type: **gacutil /u** <local directory> **\AgilentPNA835x.exe** and click **OK** to remove the assembly from the GAC.
2. On your PC, click **Start**, then **Run**, then type: **regasm /unregsiter** <local directory> **\ agilentpna835x.dll** and click **OK** to unregister the assembly.
3. To uninstall PNA Proxy.exe use the **Add/Remove Programs** utility in the control panel.

Working with Events

- What are Events?
- Using the Analyzer's Events
- Event ID's
- Filtering Events
- List of Events
- Out of Range Errors
- Troubleshooting Problems with Events

See Events Example.

Other Topics about COM Concepts

What are Events?

Windows applications work from user-initiated events such as mouse moves and mouse clicks. A mouse-click produces an event that the programmer can either ignore or "handle" by providing an appropriate subroutine like this:

```
Sub DoThis_onClick
  Perform something
 End Sub
```

If this subroutine were in your program and the mouse-click event occurs on your PC, it would generate a "Callback" to the client and interupt whatever it was doing and handle the event.

A more practical example of an event in the analyzer is Limit test. If limit test is on and the measurement fails, the analyzer produces a '"Limit-failed" event. If the measurement passed, the analyzer produces a "Limit-succeeded" event.

The Analyzer has a very sophisticated Event structure. Your program **CAN** be notified when one or more events occur. However, it may not be necessary.

For example, the analyzer has an event that will notify your program when a sweep is complete. A simpler alternative is to use a synchronous command which waits for the sweep to complete.

```
sync = True
 app.ManualTrigger sync
 chan.StartFrequency = 4.5E6
```

This would NOT work if you want the controller to do other things while waiting, like setup a power meter or sort some data. In this case you would like a "callback" from the analyzer to let your program know that the sweep has completed. For an example of this see Events Example.

Another reason to use events is when you want to be notified of several conditions when they occur, such as errors or source unlock conditions. It would not be practical to routinely poll these conditions while executing your program.

Using Events

If you decide to use the COM events to get a callback, your program must do two things:

1. Subscribe to events:

All events in the analyzer are a child of the Application object through the INetworkAnalyzerEvents Interface. You must tell the Application object that you are interested in receiving event callbacks. This process is called subscription.

In Visual Basic, this is done by including "WithEvents" in the declaration statement. The declaration below dimensions an Application object (myPNA) and subscribes to the events produced by the Application.

```
Dim WithEvents myPNA as AgilentPNA835x.Application
```

In C++, this is a bit more involved. You must queryInterface for the IconnectionPointContainer interface, locate the InetworkAnalyzerEvents interface via a call to FindConnectionPoint and call Advise().

2. Implement the Event Handler

When an event occurs, the Application object will "callback" to the client through the InetworkAnalyzerEvents interface.

In VB, click on the object window (upper left pane). Find the Application object and click it. The event interfaces will appear in the upper right pane. As you click on them, VB supplies the first line of code. You fill in the rest of the handler routine to service the event. The following is an example of a event handler subroutine.

Note: In C++, you must type the callback.

```
Private Sub OnChannelEvent( eventID as Variant, channelNumber as
Variant)
 Select Case (eventID)
  Case naEventID_CHANNEL_TRIGGER_COMPLETE:
   GetData( channelNumber )
  Case naEventID_CHANNEL_TRIGGER_ABORTED:
   MsgBox( "Hey don't touch the front panel!")
 End Select
 End Sub
```

When the trigger is complete, the application object "fires" the event by making a callback to the event handler `Sub` OnChannelEvent`()` .

Event IDs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Sev		**C**	**R**	**Facility**											**Code**																

Filtering Events

There are over 140 different events that you subscribe to when you "Dim WithEvents..." (or the equivalent in your programming language). Monitoring all of these conditions slows the speed of the analyzer significantly. The following methods allow you to filter the events so that you only monitor specific conditions.

- AllowEventMessage - monitor a specific event
- AllowAllEvents - monitor ALL events
- DisallowAllEvents - monitor NO events
- AllowEventCategory - monitor specific event categories (discussed later)
- AllowEventSeverity - monitor events having one or more of the following severity levels associated with them.

Code	Severity Enumeration
00	naEventSeveritySUCCESS - the operation completed successfully
01	naEventSeverityINFORMATIONAL - events that occur without impact on the measurement integrity
10	naEventSeverityWARNING - events that occur with potential impact on measurement integrity
11	naEventSeverityERROR - events that occur with serious impact on measurement integrity

List of Events

The following is a list of categories and the general types of events they include. Click the link view the event details.

Category Enumeration	Callback
naEventCategory_PARSER	OnSCPIEvent
naEventCategory_MEASURE	OnMeasurementEvent
naEventCategory_CHANNEL	OnChannelEvent
naEventCategory_HW	OnHardwareEvent

naEventCategory_CAL	OnCalEvent
naEventCategory_USER	OnUserEvent
naEventCategory_DISPLAY	OnDisplayEvent
naEventCategory_GENERAL	OnSystemEvent

Note: Use the MessageText Method to get a text message describing the event.

Out of Range Errors

When you attempt to set a value on an active function that is beyond the range (min or max) of the allowable values, the analyzer limits that value to an appropriate value (min or max) and sets the function to the limited value. From the front panel controls this is visually evident by the limited value in the edit box or by the annotation on the display. An example would be attempting to set the start frequency below 300kHz. The edit control doesn't allow the number to fall below 300kHz.

When the automation user programs a setting (such as start frequency below the allowable limits) the same behavior takes place. The analyzer accepts the limited value. However, in order to learn what setting took place, you have to read the HRESULT.

All automation calls return HRESULTs. By default the HRESULT returned when an overlimit occurs is S_NA_LIMIT_OUTOFRANGE. This value is a success code, meaning that bit 31 in this 32 value is 0. Programmers should check the return code from all automation calls to determine success or failure.

Some C++ macros (like SUCCEEDED(hr) or FAILED(hr)) only check bit 31. So if you are interested in trapping this outOfRange error you will have to check for S_NA_LIMIT_OUTOFRANGE explicity.

Alternatively, you can configure the analyzer to report outOfRange conditions with an error code. Use the method: App.SetFailOnOverRange (true). With this method set TRUE, any overrange error will return E_NA_LIMIT_OUTOFRANGE_ERROR.

This method is provided for the benefit of VB clients. VB users can't detect specific success codes because the VB runtime strips off the HRESULT and only raises a run time error if bit 31 is set, indicating a fail code.

Troubleshooting Problems with Callbacks

When you do callbacks, the client PC becomes the server and the analyzer (server) becomes the client. Callbacks can only take place when both server and client are in the same workgroup or in the same domain. See Configure for COM.

SCPI Command Tree

IEEE- 488.2 Common Commands

ABORt	Stops all sweeps
CALCulate __Click to hide CALC commands	
:CORRection	Sets Electrical Delay and Phase Offset
:CUSTom	Creates custom measurements
:DATA	Sends and queries data.

:FILTer	Sets time domain gating
:FORMat	Sets the display format
:FUNCtion	Controls Trace Statistics
:LIMit	Controls limit lines for pass / fail testing
:MARKer	Controls the marker settings
:MATH	Performs math on the memory trace
:NORMalize	Specifies the normalization features used for a receiver power calibration
:PARameter	Creates and deletes measurements
:RDATa?	Queries receiver data
:SMOothing	Controls point-to-point smoothing
:TRANsform	Controls time domain transform settings
CONTrol	Controls the rear-panel connectors
DISPlay	Controls the display settings
FORMat	Sets the format for data transfer
HCOPy	Controls hardcopy printing
INITiate	Sets continuous or manual triggering
MMEMory	Saves and recalls instrument states
OUTPut	Turns RF power ON and OFF

SENSe __Click to hide SENSe commands

:AVERage	Sets sweep averaging parameters
:BANDwidth	Specifies the IF filter bandwidth
:CORRection	Provides non-guided calibration capability
:CORR:COLL:CKIT	Defines calibration standards
:CORR:CSET	Manages Cal Sets
:CORR:COLL:GUID	Provides Guided calibration capability
:COUPle	Sets sweep as Chopped or Alternate
:FREQuency	Controls frequency sweep functions
:OFFSet	Sets frequency offset functions
:POWer	Sets receiver attenuation and overpower protection
:ROSCillator	Returns the source of the reference oscillator.
:SEGMent	Defines the segment sweep settings.
:SWEep	Specifies the sweep modes of the analyzer.
SOURce	Controls the power to the DUT
SOURce:POWer	Provides for Source Power Correction
STATus	Reads the analyzer status registers
SYSTem	Controls the analyzer defaults
TRIGger	Starts or ends a measurement

IEEE 488.2 Common Commands

***CLS -** Clear Status

***ESE -** Event Status Enable

***ESE?** - Event Status Enable Query

***ESR?** - Event Status Enable Register

***IDN?** - Identify

***OPC** - Operation complete command

***OPC?** - Operation complete query

***OPT?** - Identify Options Query

***RST** - Reset

***SRE** - Service Request Enable

***SRE?** - Service Request Enable Query

***STB?** - Status Byte Query

***TST?** - Result of Self-test Query

***WAI** - Wait

***CLS - Clear Status**

Clears the instrument status byte by emptying the error queue and clearing all event registers. Also cancels any preceding *OPC command or query. See Status Commands and Reading the Analyzer's Status Registers.

***ESE - Event Status Enable**

Sets bits in the standard event status enable register. See Status Commands and Reading the Analyzer's Status Registers.

***ESE? - Event Status Enable Query**

Returns the results of the standard event enable register. The register is cleared after reading it. See Status Commands and Reading the Analyzer's Status Registers.

***ESR - Event Status Enable Register**

Reads and clears event status enable register. See Status Commands and Reading the Analyzer's Status Registers.

***IDN? - Identify**

Returns a string that uniquely identifies the analyzer. The string is of the form "Agilent Technologies",<model number>,<serial "number>,<software revision>" .

***OPC - Operation complete command**

Generates the OPC message in the standard event status register when all pending overlapped operations have been completed (for example, a sweep, or a Default). See Understanding Command Synchronization.

***OPC? - Operation complete query**

Returns an ASCII "1" when all pending overlapped operations have been completed. See Understanding Command Synchronization

---

***OPT? - Identify Options Query**

Returns a string identifying the analyzer option configuration.

***RST - Reset**

Executes a device reset and cancels any pending *OPC command or query, exactly the same as a SYSTem:PRESet. The contents of the analyzer's non-volatile memory are not affected by this command.

***SRE - Service Request Enable**

Before reading a status register, bits must be enabled. This command enables bits in the service request register. The current setting is saved in non-volatile memory. See Status Commands and Reading the Analyzer's Status Registers.

***SRE? - Service Request Enable Query**

Reads the current state of the service request enable register. The register is cleared after reading it. The return value can be decoded using the table in Status Commands. See also Reading the Analyzer's Status Registers.

***STB? - Status Byte Query**

Reads the value of the instrument status byte. The register is cleared only when the registers feeding it are cleared. See Status Commands and Reading the Analyzer's Status Registers.

***TST? - Result of Self-test Query**

Returns the result of a query of the analyzer hardware status. An **0** indicates no failures found. Any other value indicates one or more of the following conditions exist. The value returned is the Weight (or sum of the Weights) of the existing conditions. For example:

- If **4** is returned from *TST?, an **Overpower** condition exists.
- If **6** is returned, both **Unleveled** and **Overpower** conditions exists.

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Phase Unlock	the source has lost phaselock. This could be caused by a reference channel open or a hardware failure.
1	2	Unleveled	the source power is unleveled. This could be a source is set for more power than it can deliver at the tuned frequency. Or it could be caused by a hardware failure.
2	4	Overpower	too much power is detected at the input. This is from either using an amplifier, or a hardware failure.
3	8	EE Write Failed	an attempted write to the EEPROM has failed. This is possibly caused by a hardware failure.
4	16	YIG Cal Failed	the analyzer was unable to calibrate the YIG. Either the phaselock has been lost or there has been a hardware failure.
5	32	Ramp Cal Failed	the analyzer was unable to calibrate the analog ramp generator due to a possible hardware failure.
6	64	OverTemp	the source temperature sensor exceeds the limit. It could result from restricted airflow or a broken fan

---

***WAI - Wait**

Prohibits the instrument from executing any new commands until all pending overlapped commands have been completed. See Understanding Command Synchronization

About Triggering

Abort Command

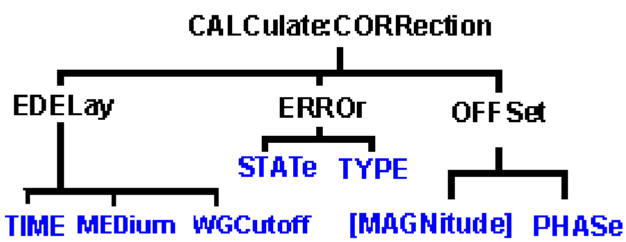
ABORt

(Write-only) Stops all sweeps - then resume per current trigger settings. This command is the same as INITtiate:IMMediate (restart) except if a channel is performing a single sweep, ABORt will stop the sweep, but not initiate another sweep.

Examples	ABOR abort
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

Calc:Correction Commands

Controls **Electrical Delay** and **Offset**



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. To select the measurement use CALC<ChanNum>:PAR:SEL <MeasName>.

CALCulate<cnum>:CORRection:EDELay:MEDium <char>

(Read-Write) Sets the media used when calculating the electrical delay.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1.
<num>	Choose from: **COAX** for coaxial medium, **WAVE**guide for

waveguide medium.

Examples CALC:CORR:EDEL:MED COAX
calc3:corr:edelay:medium waveguide

Query Syntax CALCulate<cnum>:CORRection:EDELay:MEDium?
Return Type Character

Overlapped? No
Default **COAX**

CALCulate<cnum>:CORRection:EDELay:TIME <num>

(Read-Write) Sets the electrical delay for the selected measurement. **Critical Note:**

Parameters

<cnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

<num> Electrical delay in seconds. Choose any number between**:**
-10.00 and **10.00**
Use SENS:CORR:RVEL:COAX <num> to set Velocity factor.

Examples CALC1:CORR:EDEL:TIME 1NS
calculate2:correction:time 0.5e-12

Query Syntax CALCulate:CORRection:EDELay:TIME?
Return Type Character

Overlapped? No
Default 0 seconds

CALCulate<cnum>:CORRection:EDELay:WGCutoff <num>

(Read-Write) Sets the waveguide cutoff frequency used when the electrical delay media is set to WAVEguide. (See CALCulate:CORRection:EDELay:MEDium <char>.)

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1.

<num> Waveguide cutoff frequency used with the electrical delay calculation.

Examples CALC:CORR:EDEL:WGC 18.067 GHz
calculate3:correction:edelay:wgcutoff 14.047 ghz

Query Syntax CALCulate<cnum>:CORRection:EDELay:WGCutoff?
Return Type Character

Overlapped? No
Default 45 MHz

CALC<ch>:CORRection:ERROr:TYPE <string>

(Read-Write) Set the error correction type (caltype) for this measurement. The string argument can be either the GUID of the caltype or the registered name. To determine the name or guid, see the above command (SENS:CORR:CSET:TYPE:CAT?)CALC<ch>:CORR:ERROr:TYPE? <optional enum>. This command requires a measurement be selected for the calc block (CALC:PAR:SEL).

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
Examples	CALC:CORR:ERRO:TYPE CALC2:CORRection:ERROr:TYPE?
Query Syntax	Not Applicable
Return Type	**string**
Overlapped?	No
Default	Not Applicable

CALC<ch>:CORRection: ERROr:TYPE?<opt enum>

(Read-Write) Returns the currently selected error correction type (caltype). This command requires a measurement be selected for the calc block (CALC:PAR:SEL).

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<optional enum>	**NAME:** (default) returns the string name of the caltype **GUID:** Returns the guid of the caltype
Examples	CALC:CORR:ERRO:TYPE? CALC2:CORRection:ERROr:TYPE?
Query Syntax	Not Applicable
Return Type	**string**
Overlapped?	No
Default	Not Applicable

CALC<ch>: CORRection: ERROr:STATe on | off

(Read-Write) Turns error correction on or off for this measurement. This command requires a measurement be selected for the calc block (CALC:PAR:SEL).

Note: You must also set the error correction type prior to turning on error correction.

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1

Examples CALC:CORR:ERRO:STATe?
CALC2:CORRection:ERROr:STATe on

Query Syntax Not Applicable
Return Type **string**
Overlapped? No
Default Not Applicable

CALC<ch>: CORRection: ERROr:STATe?

(Read-Write) Query Form: returns the status of correction for the selected measurement (on/off). This command requires a measurement be selected for the calc block (CALC:PAR:SEL).

Parameters

<ch> Any existing channel number. If unspecified, value is set to 1

Examples CALC:CORR:ERRO:STAT?
CALC2:CORRection:ERROr:STAT? on

Query Syntax Not Applicable
Return Type **on/off**
Overlapped? No
Default Not Applicable

CALCulate<cnum>:CORRection:OFFSet[:MAGNitude] <num>

(Read-Write) Specifies the power level to which the selected (unratioed) measurements data is to be adjusted by a Receiver Power Calibration. This command applies only when the selected measurement is of unratioed power. Critical Note:

Parameters

<cnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

<num> Cal power level in dBm. No limits are enforced on this value, but the PNA receivers themselves have maximum and minimum power specifications (that may differ between PNA models) which this value must comply with for a valid receiver power cal.

Examples CALC:CORR:OFFS 10DBM
calculate1:correction:offset:magnitude maximum

Query Syntax CALCulate<cnum>:CORRection:OFFSet[:MAGNitude]?

Return Type	Character
Overlapped?	No
Default	0dBm

CALCulate<cnum>:CORRection:OFFSet:PHASe <num>[<char>]

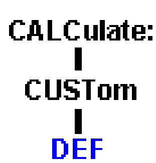
(Read-Write) Sets the phase offset for the selected measurement. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Offset phase value. Choose any number between: **-360** and **360**
<char>	Units for phase. OPTIONAL. Choose either: **DEG** - Degrees (default) **RAD** - Radians

Examples	CALC:CORR:OFFS:PHAS 10 calculate:correction:offset:phase 20rad
Query Syntax	CALCulate:CORRection:OFFSet:PHASe?
Return Type	Character, returned value always in degrees
Overlapped?	No
Default	0 degrees

Calculate:Custom Command



CALCulate<cnum>:CUST:DEF <Mname>, <ProgID>

(Write-only) Creates a custom measurement but does not display it.·

- Use DISP:WIND:STATe to create a window if it doesn't already exist.·
- Use DISP:WIND<wnum>:TRAC<tnum>:FEED<mname> to display the measurement

You must select the measurement (CALC<cnum>:PAR:SEL ,mname>) before making additional settings.

Parameters	
<cnum>	Any existing channel number. If unspecified, value is set to 1.
<Mname>	Name of the measurement. Any non-empty, unique string, enclosed in quotes.
<ProgID>	The ProgID or CLSID of the measurement to be created
Examples	CALC4:CUST:DEF 'Test', 'CustomMeasurement.1'
Query Syntax	Not applicable
Return Type	Not applicable
Overlapped?	No
Default	Not applicable

Calc:Data Commands

Controls sending and receiving data with the PNA

CALCulate:DATA
CUSTom
CATalog?

- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See Data Access Map

CALCulate<cnum>:DATA <char>,<data>

Writes Measurement data, Memory data, Normalization Divisor data, or Error terms.

CALCulate<cnum>:DATA? <char>

Reads Measurement data, Memory data, Normalization Divisor data, or Error terms.

Format of returned Measurement and Memory Data:

REAL or ASCii (see Transferring Measurement Data)

FDATA - one number per trace point
SDATA - two numbers per trace point
FMEM - one number per trace point
SMEM - two numbers per trace point

SDIV - two numbers per trace point

Format of all returned Error Terms: - two numbers per trace point

(see below for specifying <char> for error terms)

Parameters

<cnum> - Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

<char> - To write or read **Measurement (DATA), Memory (MEM)** or **Normalization Divisor (DIV)** choose from:

FDATA - formatted trace data from **measResult** location
SDATA - corrected complex trace data from **rawMeas** location
FMEM - formatted memory data from **memResult** location
SMEM - corrected complex data from **rawMemory** location
SDIV - complex data from **Normalization Divisor** location

Note: Normalization Divisor data is that obtained from a Receiver Power Calibration, for example.

<char> - To write or read **Error Terms**...

For **Response Open** calibrations:

Specify this **<char>**...	to get this **Term...**
SCORR3	Reflection Tracking

For **Response Short** calibrations:

Specify this **<char>**...	to get this **Term...**
SCORR3	Reflection Tracking

For **Response Thru** calibrations:

Specify this **<char>**...	to get this **Term...**
SCORR6	Transmission Tracking

For **Response Thru and Isolation** calibrations:

Specify this **<char>**...	to get this **Term...**
SCORR4	Isolation
SCORR6	Transmission Tracking

For **1-Port** calibrations:

Specify this **<char>**...	to get this **Term...**
SCORR1	Directivity
SCORR2	Source Match
SCORR3	Reflection Tracking

For **2-Port SOLT and TRL** calibrations

Specify this **<char>**...	to get this **Term...**
SCORR1	Forward Directivity
SCORR2	Forward Source Match
SCORR3	Forward Reflection Tracking
SCORR4	Forward Isolation
SCORR5	Forward Load Match
SCORR6	Forward Transmission Tracking
SCORR7	Reverse Directivity
SCORR8	Reverse Source Match
SCORR9	Reverse Reflection Tracking
SCORR10	Reverse Isolation
SCORR11	Reverse Load Match

SCORR12	Reverse Transmission Tracking

For **FULL 3-Port SOLT** calibrations

Specify this **<char>**...	to get this **Term...**	for this **Receiver Port .**
SCORR13	Directivity	**3** (S33)
SCORR14	Source Match	**3** (S33)
SCORR15	Reflection Tracking	**3** (S33)
SCORR16	Isolation	**3** (S31)
SCORR17	Load Match	**3** (S31)
SCORR18	Trans Tracking	**3** (S31)
SCORR19	Isolation	**1** (S13)
SCORR20	Load Match	**1** (S13)
SCORR21	Trans Tracking	**1** (S13)
SCORR22	Isolation	**3** (S32)
SCORR23	Load Match	**3** (S32)
SCORR24	Trans Tracking	**3** (S32)
SCORR25	Isolation	**2** (S23)
SCORR26	Load Match	**2** (S23)
SCORR27	Trans Tracking	**2** (S23)

EXAMPLE

```
CALC:DATA FDATA,Data(x)
 calculate2:data sdata,data(r,i)
```

See another example using this command.

Overlapped? - No
Default - Not Applicable

Notes:

- When querying memory, you must first store a trace into memory using CALC:MATH:MEMorize.
- When querying the normalization divisor, you must first store a divisor trace using CALC:NORMalize[:IMMediate].
- If normalization interpolation is ON and the number of points changes after the initial normalization, the divisor data will then be interpolated.
- When querying error terms, there must be error terms in the analyzer.
- If interpolation is ON and the number of points changes after the initial calibration, the error terms will then be the interpolated results.
- To get and put receiver data, see CALC:RDATA?
- To get uncorrected ratioed data, turn correction OFF and use Calc:Data SDATA.
- CALCulate commands act on the selected measurement. You can select one measurement in each channel. Therefore, you can have up to four measurements selected at the same time. Select the measurement for each channel using CALC:PAR:SEL.

Learn more about Error Terms

CALCulate<cnum>:DATA:CUSTom <name>,<data>

(Read-Write) Reads or writes data from a custom-named measurement buffer. Specify the measurement using CALCulate:PARameter:SELect. Critical Note:

Parameters

<cnum> Channel number of the measurement. There must be a selected

measurement on that channel. If unspecified, <cnum> is set to 1.

<name> Name of the buffer to be read or written

<data> Data to be read or written to the custom buffer. Format as one number per data point.

Examples **`CALC:DATA:CUST 'VectorResult0',0,1,2,3,4,5 'Write`**

CALC:DATA:CUST? 'VectorResult0' 'Read

Query Syntax CALCulate:DATA:CUSTom? <name>

Return Type **REAL or ASCii** (see Getting Data from the Analyzer)

Overlapped? No

Default Not Applicable

CALCulate<cnum>:DATA:CUSTom:CATalog?

(Read-only) Reads the list of buffer names (comma separated list of string values) available from the selected parameter. Specify the measurement using CALCulate:PARameter:SELect.

Critical Note:

Parameters

<cnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

Examples **`CALC:DATA:CUST:CAT?`**

calculate:data:custom:catalog?

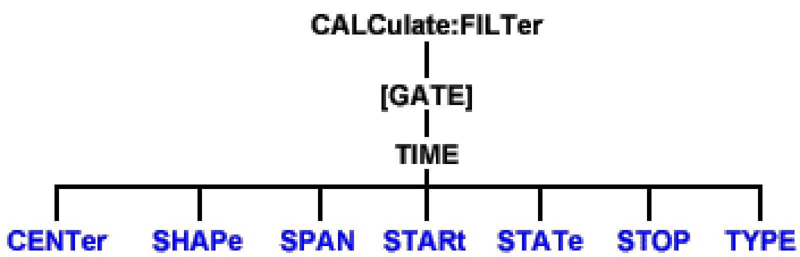
Return Type **REAL or ASCii** (see Getting Data from the Analyzer)

Overlapped? No

Default Not Applicable

Calc:Filter Commands

Controls the gating function used in time domain measurements. The gated range is specified with either (start / stop) or (center / span) commands.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.

- Learn about Gating

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:FILTer[:GATE]:TIME:CENTer <num>

(Read-Write) Sets the gate filter center time. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Center time in seconds; Choose any number between: ± (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:FILT:GATE:TIME:CENT -5 ns calculate2:filter:time:center maximum
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:CENTer?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:FILTer[:GATE]:TIME:SHAPe <char>

(Read-Write) Sets the gating filter shape when in time domain. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from **MAXimum -** the widest gate filter available **WIDE -** **NORMal -** **MINimum -** the narrowest gate filter available
Examples	CALC:FILT:GATE:TIME:SHAP MAX calculate2:filter:time:shape normal
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:SHAPe?
Return Type	Character
Overlapped?	No
Default	NORMal

CALCulate<cnum>:FILTer[:GATE]:TIME:SPAN <num>

(Read-Write) Sets the gate filter span time. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Time span in seconds; Choose any number between:

0 and 2* [(number of points-1) / frequency span]

Note: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples	CALC:FILT:GATE:TIME:SPAN 5 ns calculate2:filter:time:span maximum
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:SPAN?
Return Type	Character
Overlapped?	No
Default	20 ns

CALCulate<cnum>:FILTer[:GATE]:TIME:STATe <boolean>

(Read-Write) Turns gating state ON or OFF. **Critical Note:**

Note: Sweep type must be set to LInear Frequency in order to use Transform Gating.

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<boolean>	**ON** (or 1) - turns gating ON. **OFF** (or 0) - turns gating OFF.
Examples	CALC:FILT:TIME:STAT ON calculate2:filter:gate:time:state off
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:FILTer[:GATE]:TIME:STARt <num>

(Read-Write) Sets the gate filter start time. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Start time in seconds; any number between: **±** (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:FILT:TIME:STAR 1e-8 calculate2:filter:gate:time:start minimum
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:STARt?
Return Type	Character
Overlapped?	No
Default	10 ns

CALCulate<cnum>:FILTer[:GATE]:TIME:STOP <num>

(Read-Write) Sets the gate filter stop time. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Stop time in seconds; any number between: ± (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:FILT:TIME:STOP -1 ns calculate2:filter:gate:time:stop maximum
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME:STOP?
Return Type	Character
Overlapped?	No
Default	10 ns

CALCulate<cnum>:FILTer[:GATE]:TIME[:TYPE] <char>

(Read-Write) Sets the type of gate filter used. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from: **BPASs -** Includes (passes) the range between the start and stop times. **NOTCh** - Excludes (attenuates) the range between the start and stop times.
Examples	CALC:FILT:TIME BPAS calculate2:filter:gate:time:type notch
Query Syntax	CALCulate<cnum>:FILTer[:GATE]:TIME[:TYPE]?
Return Type	Character
Overlapped?	No
Default	BPAS

Calc:Format Command

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

- See an example using this command.
- See a List of all commands in this block.
- Learn About Data Format

CALCulate<cnum>:FORMat <char>

(Read-Write) Sets the display format for the measurement. **Critical Note:**

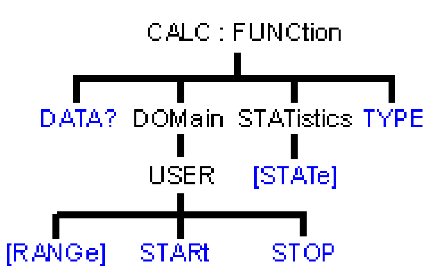
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from: • MLINear • MLOGarithmic • PHASe • IMAGinary • REAL • POLar • SMITh • SWR GDELay
Examples	CALC:FORM MLIN calculate2:format polar
Query Syntax	CALCulate<cnum>:FORMat?
Return Type	Character
Overlapped?	No
Default	MLINear

List of all commands in this block:
(Parameters in ***bold italics***)

```
:CALCulate1:FORMat MLIN
 :CALCulate1:FORMat?
```

Calc:Function Commands



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Trace Statistics

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:FUNCtion:DATA?

(Read-only) Returns the trace statistic data for the selected statistic type for the specified channel. Select the type of statistic with CALC:FUNC:TYPE. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Return Type	Character
Example	CALCulate2:FUNCtion:DATA?
Overlapped?	No
Default	Not applicable

CALCulate<cnum>:FUNCtion:DOMain:USER[:RANGe] <range>

(Read-Write) Sets the range used to calculate trace statistics. Each channel shares 10 domain ranges. The x-axis range is specified with the CALC:FUNC:DOM:USER:START and STOP commands. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<range>	Range number. Choose from: **0** to **9** **0** is Full Span of the current x-axis range **1 to 9** are user-specified ranges
Examples	CALC:FUNC:DOM:USER 4 calculate2:function:domain:user:range 0
Query Syntax	CALCulate<cnum>:FUNCtion:DOMain:USER[:RANGe]?
Return Type	Character
Overlapped?	No
Default	0 - Full Span

CALCulate<cnum>:FUNCtion:DOMain:USER:STARt <range>, <start>

(Read-Write) Sets the start of the specified user-domain range.
To apply this range, use CALC:FUNC:DOM:USER
To set the stop of the range, use CALC:FUNC:DOM:USER:STOP. **Critical Note:**

Note: This command does the same as CALC:MARK:FUNC:DOM:USER:STAR

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<range>	Range number that will receive the start value. Choose an integer between **1** and **9**

<start>	Start value of the specified range. Choose a real number between: the analyzer's **Minimum** and **Maximum** x-axis value.
Examples	CALC:FUNC:DOM:USER:STAR 1,1e9 calculate2:function:domain:user:start 2,2e9
Query Syntax	CALCulate<cnum>:FUNCtion:DOMain:USER:STARt? <range>
Return Type	Character
Overlapped?	No
Default	The analyzer's **Minimum** x-axis value

CALCulate<cnum>:FUNCtion:DOMain:USER:STOP <range>, <stop>

(Read-Write) Sets the stop of the specified user-domain range.
To apply this range, use CALC:FUNC:DOM:USER
To set the start of the range, use CALC:FUNC:DOM:USER:START

Critical Note:

Note: This command does the same as CALC:MARK:FUNC:DOM:USER:STOP
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<range>	Range number that will receive the stop value. Choose an integer between **1** and **9**
<stop>	Stop value of the specified range. Choose a real number between: the analyzer's **Minimum** and **Maximum** x-axis value.
Examples	CALC:FUNC:DOM:USER:STOP 4,5e9 calculate2:function:domain:user:stop 3,8e9
Query Syntax	CALCulate<cnum>:FUNCtion:DOMain:USER:STOP? <range>
Return Type	Character
Overlapped?	No
Default	The analyzer's **Maximum** x-axis value

CALCulate<cnum>:FUNCtion:STATistics[:STATe] <ON|OFF>

(Read-Write) Displays and hides the measurement (Trace) statistics (peak-to-peak, mean, standard deviation) on the screen.

The analyzer will display either measurement statistics or Filter Bandwidth statistics; not both.
Critical Note:
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON|OFF>	ON - Displays trace statistics OFF - Hides trace statistics
Examples	CALC:FUNC:STAT ON calculate2:function:statistics:state off
Query Syntax	CALCulate<cnum>:FUNCtion:STATistics[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)

Overlapped?	No
Default	OFF (0)

CALCulate<cnum>:FUNCtion:TYPE <char>

(Read-Write) Sets statistic TYPE that you can then query using CALC:FUNCtion:DATA?.

Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from: **PTPeak** - the difference between the max and min data points on the trace. **STDEV** - standard deviation of all data points on the trace **MEAN** - mean (average) of all data points on the trace

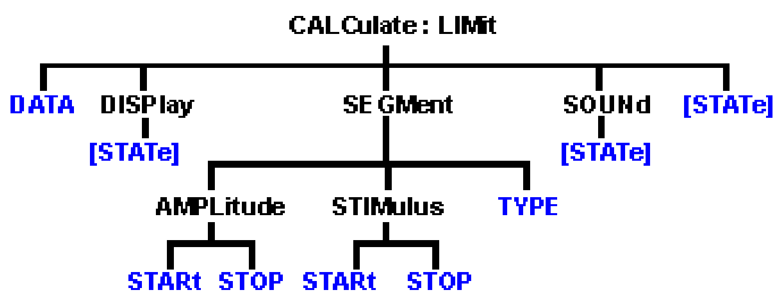
Examples	CALC:FUNC:TYPE PTP calculate2:function:type stdev

Query Syntax	CALCulate<cnum>:FUNCtion:TYPE?
Return Type	Character

Overlapped?	No
Default	PTPeak

Calc:Limit Command

Controls the limit segments used for pass / fail testing.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Limit Lines

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:LIMit:DATA <block>

(Read-Write) Sets data for limit segments. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<block>	Data for all limit segments in REAL,64 format. The following is the data format for 1 segment: **Type,BegStim, EndStim, BegResp,EndResp** **Type** Type of limit segment. Choose from 0 - Off 1 - Max 2 - Min **BegStim** Start of X-axis value (freq, power, time) **EndStim** End of X-axis value **BegResp** Y-axis value that corresponds with Start of X-axis value **EndResp** Y-axis value that corresponds with End of X-axis value

Examples	**The following writes three max limit segments for a bandpass filter.** "CALC:LIM:DATA 1,3e5,4e9,-60,0,1,4e9,7.5e9,0,0,1,7.5e9,9e9,0,-30"
Query Syntax	CALCulate<cnum>:LIMit:DATA?
Return Type	Definite length block - All 100 predefined limit segments are returned.
Overlapped?	No
Default	100 limit segments - all values set to 0

CALCulate<cnum>:LIMit:DISPlay[:STATe] <ON | OFF>

(Read-Write) Turns the display of limit segments ON or OFF (if the data trace is turned ON).

Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON** (or 1) - turns the display of limit segments ON. **OFF** (or 0) - turns the display of limit segments OFF.

Examples	CALC:LIM:DISP:STAT ON calculate2:limit:display:state off
Query Syntax	CALCulate<cnum>:LIMit:DISPlay[:STATe]?

Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

CALCulate<cnum>:LIMit:SEGMent<snum>AMPLitude:STARt <num>

(Read-Write) Sets the start (beginning) of the Y-axis amplitude (response) value. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<snum>	Segment number; if unspecified, value is set to 1.
<num>	Choose any number between: **-500** and **500** Display value is limited to the Maximum and Minimum displayed Y-axis values.
Examples	CALC:LIM:SEGM1:AMPL:STAR 10 calculate2:limit:segment2:amplitude:start 10
Query Syntax	CALCulate<cnum>:LIMit:SEGMent<snum>AMPLitude:STARt?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:LIMit:SEGMent<snum>AMPLitude:STOP <num>

(Read-Write) Sets the stop (end) of the Y-axis amplitude (response) value. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<snum>	Segment number; if unspecified, value is set to 1.
<num>	Choose any number between: **-500** and **500** Display value is limited to the Maximum and Minimum displayed Y-axis values.
Examples	CALC:LIM:SEGM1:AMPL:STOP 10 calculate2:limit:segment2:amplitude:stop 10
Query Syntax	CALCulate<cnum>:LIMit:SEGMent<snum>AMPLitude:STOP?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:LIMit:SEGMent<snum>STIMulus:STARt <num>

(Read-Write) Sets the start (beginning) of the X-axis stimulus value. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<snum>	Segment number; if unspecified, value is set to 1.
<num>	Choose any number within the X-axis span of the analyzer.

Examples	CALC:LIM:SEGM1:STIM:STAR 10 calculate2:limit:segment2:stimulus:start 10
Query Syntax	CALCulate<cnum>:LIMit:SEGMent<snum>STIMulus:STARt?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:LIMit:SEGMent<snum>STIMulus:STOP <num>

(Read-Write) Sets the stop (end) of the X-axis stimulus value. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<snum>	Segment number; if unspecified, value is set to 1.
<num>	Choose any number within the X-axis span of the analyzer.
Examples	CALC:LIM:SEGM1:AMPL:STOP 10 calculate2:limit:segment2:stimulus:stop 10
Query Syntax	CALCulate<cnum>:LIMit:SEGMent<snum>STIMulus:STOP?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:LIMit:SEGMent<snum>:TYPE <char>

(Read-Write) Sets the type of limit segment. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<snum>	Segment number. Choose any number between: **1** and **100** If unspecified, value is set to 1.
<char>	Choose from: **LMAX** - a MAX limit segment. Any response data exceeding the MAX value will fail. **LMIN** - a MIN limit segment. Any response data below the MIN value will fail. **OFF** - the limit segment (display and testing) is turned OFF.
Examples	CALC:LIM:SEGM:TYPE LMIN calculate2:limit:segment3:type lmax
Query Syntax	CALCulate<cnum>:LIMit:SEGMent<snum>:TYPE?
Return Type	Character
Overlapped?	No
Default	OFF

CALCulate<cnum>:LIMit:SOUNd[:STATe] <ON | OFF>

(Read-Write) Turns limit testing fail sound ON or OFF. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON** (or 1) - turns sound ON. **OFF** (or 0) - turns sound OFF.
Examples	CALC:LIM:SOUN ON calculate2:limit:sound:state off
Query Syntax	CALCulate<cnum>:LIMit:SOUNd[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:LIMit:STATe <ON | OFF>

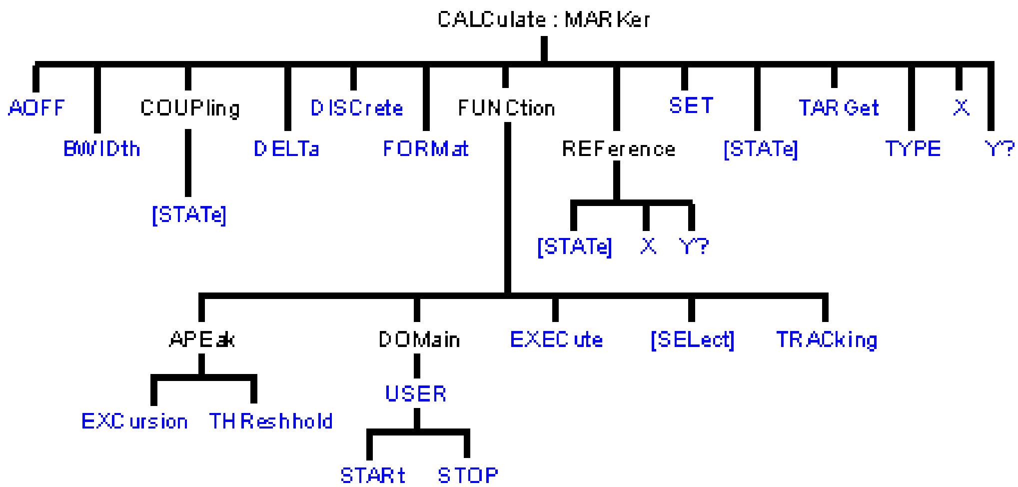
(Read-Write) Turns limit segment **testing** ON or OFF.
Use CALC:LIM:DISP to turn ON and OFF the **display** of limit segments. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON** (or 1) - turns limit testing ON. **OFF** (or 0) - turns limit testing OFF.
Examples	CALC:LIM:STAT ON calculate2:limit:state off
Query Syntax	CALCulate<cnum>:LIMit:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

Calc:Marker Commands

Controls the marker settings used to remotely output specific data to the computer.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See commands for controlling the marker readout number and size
- Learn about Markers

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

Note: The Reference Marker is Marker Number 10

CALCulate<cnum>:MARKer:AOFF

(Write-only) Turns all markers off for selected measurement.

Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Examples	CALC:MARK:AOFF calculate2:marker:aoff
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

CALCulate<cnum>:MARKer:BWIDth <num>

(Read-Write) Turns on and sets markers 1 through 4 to calculate filter bandwidth. The <num> parameter sets the value below the maximum bandwidth peak that establishes the bandwidth of a filter. For example, if you want to determine the filter bandwidth 3 db below the bandpass peak value, set <num> to -3.

This feature activates markers 1 through 4. To turn off these markers, either turn them off individually or turn them All Off.

The analyzer screen will show either Bandwidth statistics OR Trace statistics; not both.

To search a User Range with the bandwidth search, first activate marker 1 and set the desired User Range. Then send the CALC:MARK:BWID command. The user range used with bandwidth search only applies to marker 1 searching for the max value. The other markers

may fall outside the user range.
Critical Note:
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Target value below filter peak. Choose any number between: **-500** and **500**

Examples	CALC:MARK:BWID -3 calculate2:marker:bwidth -2.513
Query Syntax	CALCulate<cnum>:MARKer:BWIDth? Returns the results of bandwith search:
Return Type	Four Character values separated by commas: bandwidth, center Frequency, Q, loss.
Overlapped?	No
Default	-3

CALCulate<cnum>:MARKer<mkr>:COUPling[:STATe]<ON|OFF>

(Read-Write) Sets and Reads the state of Coupled Markers (ON and OFF) **Critical Note:**
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any existing marker number from 1 to 10; if unspecified, value is set to 1.
<ON|OFF>	**False (0**) - Turns Coupled Markers OFF **True (1)** - Turns Coupled Markers ON

Examples	CALC:MARK:COUP ON calculate2:marker8:coupling off
Query Syntax	CALCulate<cnum>:MARKer<mkr>:COUPling:[STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:MARKer<mkr>:DELTa <ON|OFF>

(Read-Write) Specifies whether marker is relative to the Reference marker or absolute.

Note: The reference marker must already be turned ON with CALC:MARK:REF:STATE.

Critical Note:
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any existing marker number from 1 to 10; if unspecified, value is set to 1.
<ON|OFF>	**ON** (or 1) - Specified marker is a Delta marker **OFF** (or 0) - Specified marker is an ABSOLUTE marker

Examples	CALC:MARK:DELT ON calculate2:marker8:delta off
Query Syntax	CALCulate<cnum>:MARKer<mkr>:DELTa?
Return Type	Boolean (1 = ON, 0 = OFF)

Overlapped?	No
Default	OFF

CALCulate<cnum>:MARKer<mkr>:DISCrete <ON|OFF>

(Read-Write) Makes the specified marker display either a calculated value between data points (interpolated data) or the actual data points (discrete data). **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any existing marker number from 1 to 10; if unspecified, value is set to 1.
<ON|OFF>	**ON** (or 1) - Specified marker displays the actual data points **OFF** (or 0) - Specified marker displays calculated data between the actual data points.
Examples	CALC:MARK:DISC ON calculate2:marker8:discrete off
Query Syntax	CALCulate<cnum>:MARKer<mkr>:DISCrete?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:MARKer<mkr>:FORMat <char>

(Read-Write) Sets the format of the data that will be returned in a marker data query CALC:MARK:Y? and the displayed value of the marker readout. The selection does not have to be the same as the measurement's display format. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<char>	Choose from: **DEFault** - The format of the selected measurement **MLINear** - Linear magnitude **MLOGarithmic** - Logarithmic magnitude **IMPedance** - (R+jX) **ADMittance** - (G+jB) **PHASe** - Phase **IMAGinary** - Imaginary part (Im) **REAL** - Real part (Re)l **POLar** - (Re, Im) **GDELay** - Group Delay **LINPhase -** Linear Magnitude and Phase **LOGPhase -** Log Magnitude and Phase
Examples	CALC:MARK:FORMat MLIN calculate2:marker8:format Character
Query Syntax	CALCulate<cnum>:MARKer<mkr>:FORMat?
Return Type	Character
Overlapped?	No
Default	DEFault

CALCulate<cnum>:MARKer<mkr>:FUNCtion:APEak:EXCursion <num>

(Read-Write) Sets amplitude peak excursion for the specified marker. The Excursion value determines what is considered a "peak". This command applies to marker peak searches (Next peak, Peak Right, Peak Left). **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any existing marker number from 1 to 10; if unspecified, value is set to 1.
<num>	Excursion value. Choose any number between **-500** and **500.** **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples CALC:MARK:FUNC:APE:EXC 10
calculate2:marker8:function:apeak:excursion maximum

Query Syntax CALCulate<cnum>:MARKer<mkr>:FUNCtion:APEak:EXCursion?
Return Type Character

Overlapped? No
Default 3

CALCulate<cnum>:MARKer<mkr>:FUNCtion:APEak:THReshold <num>

(Read-Write) Sets peak threshold for the specified marker. If a peak (using the criteria set with :EXCursion) is below this reference value, it will not be considered when searching for peaks. This command applies to marker peak searches (Next peak, Peak Right, Peak Left). **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<num>	Threshold value.Choose any number between **-500** and **500.** **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples CALC:MARK:FUNC:APE:THR -40
calculate2:marker8:function:apeak:threshold -55

Query Syntax CALCulate<cnum>:MARKer<mkr>:FUNCtion:APEak:THReshold?
Return Type Character

Overlapped? No
Default -100

CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER <range>

(Read-Write) Assigns the specified marker to a range number. The x-axis travel of the marker is constrained to the range's span. The span is specified with the CALC:MARK:FUNC:DOM:USER:START and STOP commands, unless range 0 is specified which is the full span of the analyzer.

Each channel shares 10 domain ranges. (Trace statistics use the same ranges.) More than one marker can use a domain range. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<span>	User span. Choose any Integer from **0 to 9**.

	0 is Full Span of the analyzer **1 to 9** are available for user-defined x-axis span
Examples	CALC:MARK:FUNC:DOM:USER 1 calculate2:marker8:function:domain:user 1
Query Syntax	CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER? Returns the user span number that the specified marker is assigned to.
Return Type	Character
Overlapped?	No
Default	0 - Full Span

CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER:STARt <start>

(Read-Write) Sets the start of the span that the specified marker's x-axis span will be constrained to.
Use CALC:MARK:FUNC:DOM:USER<range> to set range number
Use CALC:MARK:FUNC:DOM:USER:STOP to set the stop value.

Note: If the marker is assigned to range 0 (full span), the USER:STARt and STOP commands generate an error. You cannot set the STARt and STOP values for "Full Span".

Note: This command does the same as CALC:FUNC:DOM:USER:STAR

Critical Note:

Parameters	
<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<start>	The analyzer's **Minimum** x-axis value
Examples	CALC:MARK:FUNC:DOM:USER:START 500E6 calculate2:marker8:function:domain:user:start 1e12
Query Syntax	CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER:STARt?
Return Type	Character
Overlapped?	No
Default	The analyzer's **Minimum** x-axis value

CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER:STOP <stop>

(Read-Write) Sets the stop of the span that the marker's x-axis travel will be constrained to.
Use CALC:MARK:FUNC:DOM:USER<range> to set range number
Use CALC:MARK:FUNC:DOM:USER:STARt to set the stop value.

Note: If the marker is assigned to range 0 (full span), the USER:STARt and STOP commands generate an error. You cannot set the STARt and STOP values for "Full Span".

Note: This command does the same as CALC:FUNC:DOM:USER:STOP

Critical Note:

Parameters	
<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<stop>	Stop value of x-axis span; Choose any number between the analyzer's **MINimum** and **MAXimum** x-axis value.
Examples	CALC:MARK:FUNC:DOM:USER:STOP 500e6

calculate2:marker8:function:domain1:user:stop 1e12

Query Syntax CALCulate<cnum>:MARKer<mkr>:FUNCtion:DOMain:USER:STOP?

Return Type Character

Overlapped? No

Default The analyzer's **MAXimum** x-axis value.

CALCulate<cnum>:MARKer<mkr>:FUNCtion:EXECute [<func>]

(Write-only) Immediately executes (performs) the specified search function. If no function is specified, executes the selected function. Select the function with CALC:MARK:FUNCtion:SEL.

Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<func>	Optional argument. The function that is to be performed. Choose from: **MAXimum** - finds the highest value **MINimum** - finds the lowest value **RPEak** - finds the next valid peak to the right **LPEak** - finds the next valid peak to the left **NPEak** - finds the **next highest** value among the valid peaks **TARGet -** finds the target value to the right, wraps around to the left **LTARget** - finds the next target value to the left of the marker **RTARget** - finds the next target value to the right of the marker

Examples CALC:MARK:FUNC:EXEC
calculate2:marker2:function:execute maximum

Query Syntax Not applicable

Overlapped? No

Default Not applicable

CALCulate<cnum>:MARKer<mkr>:FUNCtion[:SELect] <char>

(Read-Write) Sets the search function that the specified marker will perform when executed. To execute (or perform) the function, use:

CALC:MARK:FUNC:EXEC **or**

CALC:MARK:FUNC:TRAC ON to automatically execute the search every sweep. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<char>	Marker function. Choose from: **MAXimum** - finds the highest value **MINimum** - finds the lowest value **RPEak** - finds the next valid peak to the right **LPEak** - finds the next valid peak to the left **NPEak** - finds the **next highest** value among the valid peaks **TARGet -** finds the target value to the right; wraps around to the left **LTARget** - finds the next target value to the left of the marker

RTARget - finds the next target value to the right of the marker

Examples	CALC:MARK:FUNC MAX calculate2:marker8:function:select ltarget
Query Syntax	CALCulate<cnum>:MARKer<mkr>:FUNCtion[:SELect]?
Overlapped?	No
Default	MAX

CALCulate<cnum>:MARKer<mkr>:TARGet <num>

(Read-Write) Sets the target value for the specified marker when doing Target Searches (CALC:MARK:FUNC:SEL <TARGet | RTARget | LTARget> **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<num>	Target value to search for; Units are NOT allowed.
Examples	CALC:MARK:TARG 2.5 calculate2:marker8:target -10.3
Query Syntax	CALCulate<cnum>:MARKer<mkr>:TARGet?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:MARKer<mkr>:FUNCtion:TRACking <ON | OFF>

(Read-Write) Sets the tracking capability for the specified marker. The tracking function finds the selected search function every sweep. In effect, turning Tracking ON is the same as doing a CALC:MARK:FUNC:EXECute command every sweep. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<ON | OFF>	**ON** (or 1) - The specified marker will "Track" (find) the selected function every sweep. **OFF** (or 0) - The specified marker will find the selected function **only** when the CALC:MARK:FUNC:EXECute command is sent.
Examples	CALC:MARK:FUNC:TRAC ON calculate2:marker8:function:tracking off
Query Syntax	CALCulate<cnum>:MARKer<mkr>:FUNCtion:TRACking?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:MARKer:REFerence[:STATe] <ON | OFF>

(Read-Write) Turns the reference marker (marker 10) ON or OFF. When turned OFF, existing Delta markers revert to absolute markers. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected

	measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON** (or 1) - turns reference marker ON **OFF** (or 0) - turns reference marker ON
Examples	CALC:MARK:REF ON calculate2:marker:reference:state OFF
Query Syntax	CALCulate<cnum>:MARKer:REFerence[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:MARKer:REFerence:X <num>

(Read-Write) Sets and returns the absolute x-axis value of the reference marker (marker 10).
Critical Note:

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	X-axis value. Choose any number within the operating domain of the reference marker.
Examples	`CALC:MARK:REF:X 1e9` calculate2:marker:reference:x 1e6
Query Syntax	CALCulate<cnum>:MARKer:REFerence:X?
Return Type	Character
Overlapped?	No
Default	If the first Marker, turns ON in the middle of the X-axis span. If not, turns ON at the position of the active marker.

CALCulate<cnum>:MARKer:REFerence:Y?

(Read-only) Returns the absolute Y-axis value of the reference marker. **Critical Note:**
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Examples	`CALC:MARK:REF:Y?` calculate2:marker:reference:y?
Return Type	Character
Overlapped?	No
Default	Not applicable

CALCulate<cnum>:MARKer<mkr>:TYPE <char>

(Read-Write) Sets the type of the specified marker. **Critical Note:**
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<char>	Choose from: **NORMal** - a marker that stays on the assigned X-axis position unless moved or searching. **FIXed** - a marker that will not leave the assigned X or current Y-axis position.

Examples	CALC:MARK:TYPE NORM calculate2:marker2:type fixed
Query Syntax	CALCulate<cnum>:MARKer<mkr>:TYPE?
Return Type	Character
Overlapped?	No
Default	NORMal

CALCulate<cnum>:MARKer<mkr>:SET <char>

(Read-Write) Sets the selected instrument setting to assume the value of the specified marker.
Critical Note:
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1
<char>	Choose from: **CENTer -** changes center frequency to the value of the marker **SPAN -** changes the sweep span to the span that is defined by the delta marker and the marker that it references. Unavailable if there is no delta marker. **STARt -** changes the start frequency to the value of the marker **STOP -** changes the stop frequency to the value of the marker **RLEVel** - changes the reference level to the value of the marker **DELay** - changes the line length at the receiver input to the phase slope at the active marker stimulus position.

Examples	CALC:MARK:SET CENT calculate2:marker8:set span
Query Syntax	CALCulate<cnum>:MARKer<mkr>:SET?
Return Type	Character
Overlapped?	No
Default	Not applicable

CALCulate<cnum>:MARKer<mkr>[:STATe] <ON|OFF>

(Read-Write) Turns the specified marker ON or OFF. **Marker 10 is the Reference Marker**. To turn all markers off, use CALC:MARK:AOFF. **Critical Note:**
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<ON|OFF>	**ON** (or 1) - turns marker ON. **OFF** (or 0) - turns marker OFF.

Examples	CALC:MARK ON calculate2:marker8 on
Query Syntax	CALCulate<cnum>:MARKer<mkr>:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	Off

CALCulate<cnum>:MARKer<mkr>:X <num>

(Read-Write) Sets the marker's X-axis value (frequency, power, or time). If the marker is set as delta, the SET and QUERY data is relative to the reference marker. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
<num>	Any X-axis position within the measurement span of the marker. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:MARK:X 100Mhz calculate2:marker8:x maximum
Query Syntax	CALCulate<cnum>:MARKer<mkr>:X?
Return Type	Character
Overlapped?	No
Default	First Marker turns ON in the middle of the X-axis span. Subsequent markers turn ON at the position of the active marker.

CALCulate<cnum>:MARKer<mkr>:Y?

(Read-only) Reads the marker's Y-axis value. The format of the value depends on the current CALC:MARKER:FORMAT setting. If the marker is set as delta, the data is relative to the reference marker. The query always returns two numbers:

- Smith and Polar formats - (Real, Imaginary)
- LINPhase and LOGPhase - (Real, Imaginary)
- All other formats - (Value,0)

Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<mkr>	Any marker number from 1 to 10; if unspecified, value is set to 1.
Examples	CALC:MARK:Y? calculate2:marker3:y?
Query Syntax	CALCulate<cnum>:MARKer<mkr>:Y?
Return Type	Character
Overlapped?	No
Default	Not applicable

Calc:Math Command

Controls math operations on the currently selected measurement and memory.

CALCulate : MATH

MEMorize FUNCtion

- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Math Operations

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:MATH:FUNCtion <char>

(Read-Write) Sets math operations on the currently selected measurement and the trace stored in memory. (There MUST be a trace stored in Memory. See CALC:MATH MEM) **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	The math operation to be applied. Choose from the following:

NORMal	Trace data only
ADD	Data + Memory
SUBTract	Data - Memory
MULTiply	Data * Memory
DIVide	Data / Memory

Examples	CALC:MATH:FUNC NORM calculate2:math:function subtract
Query Syntax	CALCulate<cnum>:MATH:FUNCtion?
Return Type	Character
Overlapped?	No
Default	NORMal

CALCulate<cnum>:MATH:MEMorize

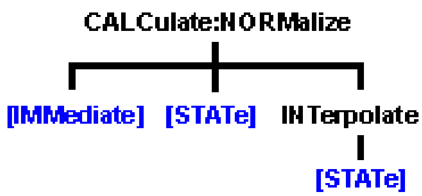
(Write-only) Puts the currently selected measurement trace into memory. (Data-> Memory)
Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Examples	CALC:MATH:MEM calculate2:math:memorize
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

Calc:Normalize Commands

Specifies the normalization features used for a receiver power calibration.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Receiver Cal

Save and recall your receiver power calibration (which use .CST file commands):

- SENS:CORR:CSET:SAVE
- SENS:CORR:CSET[:SEL]

Or use these two commands and specify either .STA or .CST file extensions:

- MMEM:LOAD
- MMEM:STOR

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:NORMalize[:IMMediate]

(Read-Write) Stores the selected measurement's data to that measurement's "divisor" buffer for use by the Normalization data processing algorithm. This command is not compatible with ratioed measurements such as S-parameters. It is intended for receiver power calibration when the selected measurement is of an unratioed power type. Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Examples	CALC:NORM calculate1:normalize:immediate
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

CALCulate<cnum>:NORMalize:STATe <ON | OFF>

(Read-Write) Specifies whether or not normalization is applied to the measurement.

Normalization is enabled only for measurements of unratioed power where it serves as a receiver power calibration. Critical Note:

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON (or 1)** - normalization is applied to the measurement. **OFF (or 0)** – normalization is NOT applied to the measurement.

Examples	CALC:NORM:STAT ON calculate2:normalize:state off
Query Syntax	CALCulate<cnum>:NORMalize:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:NORMalize:INTerpolate[:STATe] <ON | OFF>

(Read-Write) Turns normalization interpolation ON or OFF. Normalization is enabled only for measurements of unratioed power, where it serves as a receiver power calibration. Critical Note:

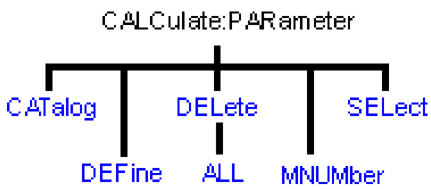
Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON | OFF>	**ON (or 1**) – turns interpolation ON. **OFF (or 0)** – turns interpolation OFF.

Examples	CALC:NORM:INT ON calculate2:normalize:interpolate:state off
Query Syntax	CALCulate<cnum>:NORMalize:INTerpolate[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

Calc:Parameter Commands

Lists, creates, selects and deletes measurements



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Measurement Parameters

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:PARameter:CATalog?

(Read-only) Returns the names and parameters of existing measurements for the specified channel. **Critical Note:**

Parameters

<cnum>	Channel number of the measurements to be listed. If unspecified, <cnum> is set to 1.
Examples	CALC:PAR:CAT? calculate2:parameter:catalog?
Query Syntax	CALCulate<cnum>:PARameter:CATalog?
Return Type	String - "<measurement name>,<parameter>,[<measurement name>,<parameter>...]"
Overlapped?	No
Default	"CH1_S11_1,S11"

CALCulate<cnum>:PARameter:DEFine <Mname>,<param>[,load]

(Write-only) Creates a measurement but does NOT display it.

- Use DISP:WIND:STATe to create a window if it doesn't already exist.
- Use DISP:WIND<wnum>:TRAC<tnum>:FEED <Mname> to display the measurement.

You must select the measurement (CALC<cnum>:PAR:SEL <mname>) before making additional settings. **Critical Note:**

Parameters

<cnum>	Channel number of the new measurement. Choose any number between: **1** and **4** If unspecified, value is set to 1.
<Mname>	Name of the measurement. Any non-empty, unique string, enclosed in quotes.
<param>	Parameter Choose from the following for S-Parameter measurements **S11 I S22 I S12 I S21** For 3-port analyzers only: **S33 I S13 I S31 I S23 I S32**

For the following non S-Parameter measurements, Specify the source port with: SENSe:SWEep:SRCPort <1|2>)

Choose from the following for non-ratioed measurements:
A | B | C | R1 | R2

Choose from the following for ratioed measurements:

<param>	**Description**
AB	A/B
AC	A/C - 3 port analyzers only
BA	B/A
BC	B/C - 3 port analyzers only
CA	C/A - 3 port analyzers only
CB	C/B - 3 port analyzers only
AR1	A/R1
BR1	B/R1
CR1	C/R1 - 3 port analyzers only
AR2	A/R2
BR2	B/R2
R1A	R1/A
R2A	R2/A
R1B	R1/B
R2B	R2/B
R1C	R1/C - 3 port analyzers only
R2R1	R2/R1
R1R2	R1/R2

[load] Optional argument; specifies the device port which will provide the load for the measurement (Multi-port reflection measurements only). This argument is ignored if a transmission S-parameter is specified.)

Examples

```
CALC:PAR:DEF 'Test',S12
calculate2:parameter:define 'test',s22
```

CALC4:PAR:DEF 'ch4_S33',S33,2 'Defines an S33 measurement with a load on port2 of the analyzer.

Query Syntax Not Applicable; see Calc:Par:Cat?

Overlapped? No
Default Not Applicable

CALCulate<cnum>:PARameter:DELete [:NAME]<Mname>

(Write-only) Deletes the specified measurement. **Critical Note:**

Parameters

<cnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

<Mname> String - Name of the measurement

Examples CALC:PAR:DEL 'TEST'
calculate2:parameter:delete 'test'

Query Syntax Not Applicable

Overlapped? No
Default Not Applicable

CALCulate<cnum>:PARameter:DELete:ALL

(Write-only) Deletes all specified measurements. **Critical Note:**

Parameters

<cnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.

Examples CALC:PAR:DEL:ALL
calculate2:parameter:delete:all

Query Syntax Not Applicable

Overlapped? No
Default Not Applicable

CALCulate<cnum>:PARameter:MNUMber?

(Read-only) Returns the measurement number of the selected measurement. This is useful when needing to identify a measurement by number, such as with Status:Ques:Lim or Status:Oper:Aver commands. **Critical Note:**

Parameters

<cnum> Channel number of the measurement. If unspecified, <cnum> is set to 1.

Examples CALC:PAR:MNUM?
calculate2:parameter:mnumber?

Query Syntax CALCulate<cnum>:PARameter:MNUMber?
Return Type Integer

Overlapped? No
Default Not Applicable

CALCulate<>:PARameter:MNUMber <>

(Read-Write)

Parameters

<> .
<>

Examples

Query Syntax

Return Type

Overlapped?

Default

CALCulate<cnum>:PARameter:SELect <Mname>

(Read-Write) Sets the selected measurement. Most CALC: commands require that this command be sent before a setting change is made. One measurement on each channel can be selected at the same time. To obtain a list of currently named measurements, use CALC:PAR:CAT? **Critical Note:**

Parameters

<cnum>	Channel number of the measurement to be selected. If unspecified, <cnum> is set to 1.
<Mname>	String - Name of the measurement. (Do NOT include the parameter name.)
Examples	CALC:PAR:SEL 'TEST' calculate2:parameter:select 'test'
Query Syntax	CALCulate:PARameter:SELect?
Return Type	String
Overlapped?	No
Default	No Selection

Calc:RData Command

Generally when you query the analyzer for data, you expect that the number of data values returned will be consistent with the number of points in the sweep.

However, if you query **receiver** data while the instrument is sweeping, the returned values may contain zeros. For example, if your request for receiver data is handled on the 45th point of a 201 point sweep, the first 45 values will be valid data, and the remainder will contain complex zero.

This can be avoided by synchronizing this request with the end of a sweep or putting the channel in hold mode.

Learn about Unratioed Measurements

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:RDATA? <char>

(Read-only) Returns receiver data for the selected measurement. To query measurement data, see CALC:DATA? **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from receivers: **A** **B** **R1** **R2** **REF -** returns either R1 or R2 data depending on the source port of the CALC-selected measurement.
Example	GPIB.Write "INITiate:CONTinuous OFF" GPIB.Write "INITiate:IMMediate;*wai"

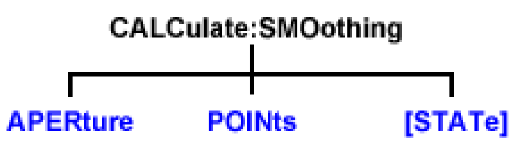
	GPIB.Write "CALCulate:RDATA? A"
Return Type	Character - Two numbers per data point
Overlapped?	No
Default	Not Applicable

List of all commands in this block:
(Parameters in ***bold italics***)

```
:CALCulate1:RDATA? A
```

Calc:Smoothing Commands

Controls point-to-point smoothing. Smoothing is a noise reduction technique that averages adjacent data points in a measurement trace. Choose the amount of smoothing by specifying either the number of points or the aperture. Smoothing is not the same as CALC:AVERage which averages each data point over a number of sweeps.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands.
- Learn about Smoothing

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:SMOothing:APERture <num>

(Read-Write) Sets the amount of smoothing as a percentage of the number of data points in the channel. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Percentage value. Choose any number between: **1** and **25**
Examples	CALC:SMO:APER 2 calculate2:smoothing:aperture 20.7
Query Syntax	CALCulate<cnum>:SMOothing:APERture?
Return Type	Character
Overlapped?	No

Default	1.5

CALCulate<cnum>:SMOothing:POINts <num>

(Read-Write) Sets the number of adjacent data points to average. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Number of points from 1 point to maximum of 25% of data points in the channel. For example: if number of points in a data trace = 401, the maximum value for points = 100. The points value is always rounded to the closest odd number.
Examples	CALC:SMO:POIN 50 calculate2:smoothing:points 21
Query Syntax	CALCulate<cnum>:SMOothing:POINts?
Return Type	Character
Overlapped?	No
Default	3

CALCulate<cnum>:SMOothing[:STATe] <ON | OFF>

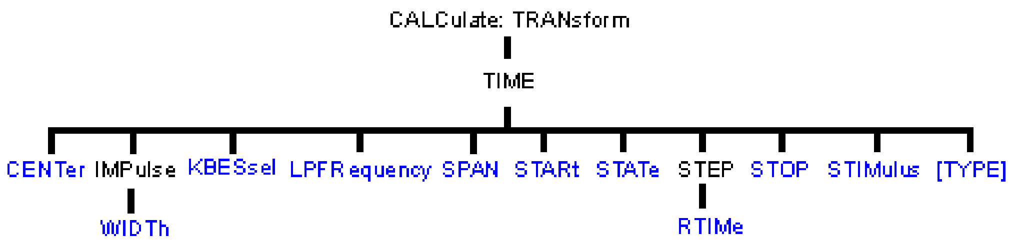
(Read-Write) Turns data smoothing ON or OFF. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON	OFF>	**ON** (or 1) - turns smoothing ON. **OFF** (or 0) - turns smoothing OFF.
Examples	CALC:SMO ON calculate2:smoothing:state off
Query Syntax	CALCulate<cnum>:SMOothing[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

Calc:Transform Commands

Specifies the settings for time domain transform.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Time Domain

Note: CALCulate commands act on the selected measurement. You can select one measurement in each channel. Select the measurement for each channel using CALC:PAR:SEL.

CALCulate<cnum>:TRANsform:TIME:CENTer <num>

(Read-Write) Sets the center time for time domain measurements. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Center time in seconds; any number between: ± (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:TRAN:TIME:CENT 1e-8 calculate2:transform:time:center 15 ps
Query Syntax	CALCulate<cnum>:TRANsform:TIME:CENTer?
Return Type	Character
Overlapped?	No
Default	0

CALCulate<cnum>:TRANsform:TIME:IMPulse:WIDTh <num>

(Read-Write) Sets the impulse width for the transform window. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Impulse width in seconds; Choose any number between: **.6 / frequency span** and **1.39 / frequency span**
Examples	CALC:TRAN:TIME:IMP:WIDTh 10 calculate2:transform:time:impulse:width 13
Query Syntax	CALCulate<cnum>:TRANsform:TIME:IMPulse:WIDTh?
Return Type	Character
Overlapped?	No
Default	.98 / Default Span

CALCulate<cnum>:TRANsform:TIME:KBESsel <num>

(Read-Write) Sets the parametric window for the Kaiser Bessel window. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Window width for Kaiser Bessel in seconds; Choose any number between: **0.0** and **13.0**
Examples	CALC:TRAN:TIME:KBES 10 calculate2:transform:time:kbessel 13
Query Syntax	CALCulate<cnum>:TRANsform:TIME:KBESsel?
Return Type	Character
Overlapped?	No
Default	6

CALCulate<cnum>:TRANsform:TIME:LPFREQuency

(Write-only) Sets the start frequencies in LowPass Mode. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
Examples	CALC:TRAN:TIME:LPFR calculate2:transform:time:lpfrequency
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

CALCulate<cnum>:TRANsform:TIME:SPAN <num>

(Read-Write) Sets the span time for time domain measurements. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Span time in seconds; any number between: **0** and 2* [(number of points-1) / frequency span] **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:TRAN:TIME:SPAN 1e-8 calculate2:transform:time:span maximum
Query Syntax	CALCulate<cnum>:TRANsform:TIME:SPAN?
Return Type	Character
Overlapped?	No
Default	20 ns

CALCulate<cnum>:TRANsform:TIME:STARt <num>

(Read-Write) Sets the start time for time domain measurements. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Start time in seconds; any number between: ± (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:TRAN:TIME:STAR 1e-8 calculate2:transform:time:start minimum
Query Syntax	CALCulate<cnum>:TRANsform:TIME:STARt?
Return Type	Character
Overlapped?	No
Default	-10 ns

CALCulate<cnum>:TRANsform:TIME:STATe <ON | OFF>

(Read-Write) Turns the time domain transform capability ON or OFF. **Critical Note:**

Note: Sweep type must be set to Linear Frequency in order to use Time Domain Transform.

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<ON|OFF>	**ON** (or 1) - turns time domain ON. **OFF** (or 0) - turns time domain OFF.
Examples	CALC:TRAN:TIME:STAT ON calculate2:transform:time:state off
Query Syntax	CALCulate<cnum>:TRANsform:TIME:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

CALCulate<cnum>:TRANsform:TIME:STOP <num>

(Read-Write) Sets the stop time for time domain measurements. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Stop time in seconds; any number between: ± (number of points-1) / frequency span **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	CALC:TRAN:TIME:STOP 1e-8 calculate2:transform:time:stop maximum

Query Syntax	CALCulate<cnum>:TRANsform:TIME:STOP?
Return Type	Character
Overlapped?	No
Default	10 ns

CALCulate<cnum>:TRANsform:TIME:STEP:RTIMe <num>

(Read-Write) Sets the step rise time for the transform window. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<num>	Rise time in seconds; Choose any number between: **.45 / frequency span** and **1.48 / frequency span**
Examples	CALC:TRAN:TIME:STEP:RTIM 1e-8 calculate2:transform:time:step:rtime 15 ps
Query Syntax	CALCulate<cnum>:TRANsform:TIME:STEP:RTIMe?
Return Type	Character
Overlapped?	No
Default	.99 / Default Span

CALCulate<cnum>:TRANsform:TIME:STIMulus <char>

(Read-Write) Sets the type of simulated stimulus that will be incident on the DUT. **Critical Note:**

Parameters

<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from: **STEP -** simulates a step DUT stimulus **IMPulse -** simulates a pulse DUT stimulus STEP can ONLY be used when CALC:TRAN:TIME:TYPE is set to LPASs (Lowpass). (STEP **cannot** be used with TYPE = BPASs.) **:STIM STEP** will set :TYPE to **LPASs** **:TYPE BPASs** will set :STIM to **IMPulse**
Examples	CALC:TRAN:TIME:STIM STEP calculate2:transform:time:stimulus impulse
Query Syntax	CALCulate<cnum>:TRANsform:TIME:STIMulus?
Return Type	Character
Overlapped?	No
Default	IMPulse

CALCulate<cnum>:TRANsform:TIME[:TYPE] <char>

(Read-Write) Sets the type of time domain measurement. **Critical Note:**

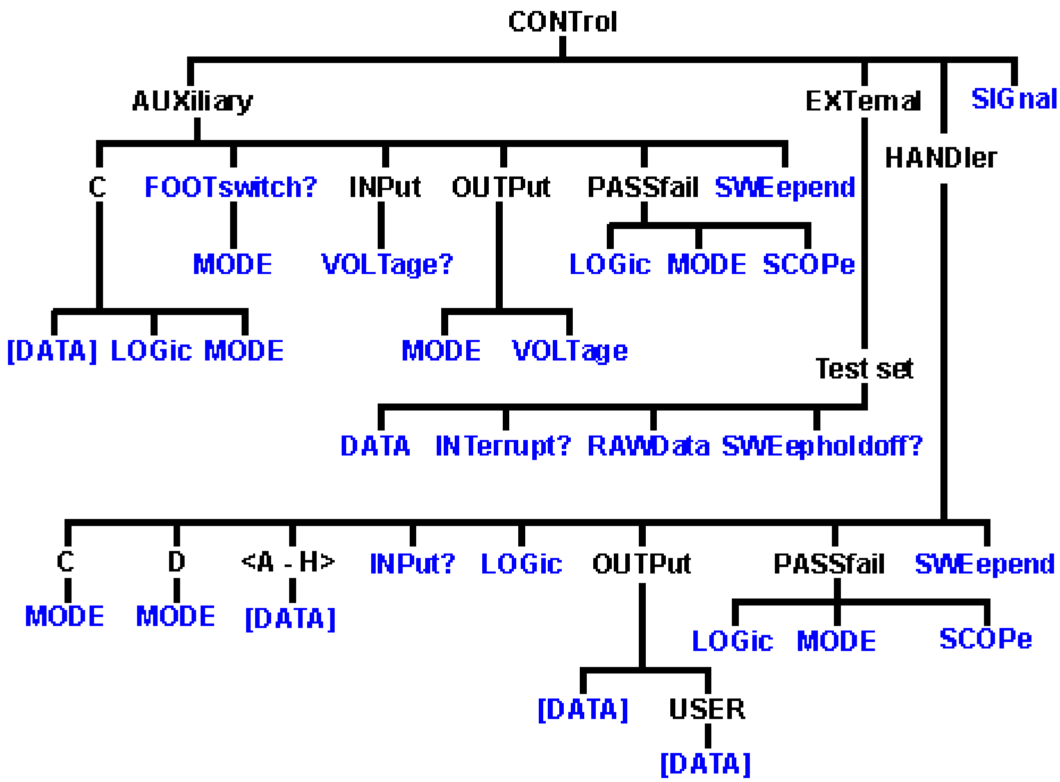
Parameters

<cnum>	Channel number of the measurement. There must be a selected

	measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Type of measurement. Choose from: **LPASs** - Lowpass; Must also send CALC:TRAN:TIME:LPFRequency before calibrating. **BPASs** - Bandpass;
	BPASs can **only** be used when CALC:TRAN:TIME:STIM is set to IMPulse. (BPASs **cannot** be used with :STIM = STEP)
	:STIM **STEP** will set :TYPE to **LPASs** :TYPE **BPASs** will set :STIM to **IMPulse**
Examples	CALC:TRAN:TIME LPAS calculate2:transform:time:type bpas
Query Syntax	CALCulate<cnum>:TRANsform:TIME[:TYPE]?
Return Type	Character
Overlapped?	No
Default	BPAS

Control Commands

Specifies the settings to remotely control the rear panel connectors.



- Click on a **blue** keyword to view the command details.
- See a List of all SCPI commands.
- See a pinout and detailed description of the rear panel connectors:
  - Auxilliary IO connector
  - External Test Set IO connector
  - Material Handler IO connector

CONTrol:AUXiliary:C[:DATA] <num>

(Read-Write) Reads and writes a 4-bit value to Port C on the Aux I/O connector. This port is connected internally to the Handler IO connector. Therefore this command will also affect the state of Port C on the Handler IO

Parameters

<num>	Data value. Choose any number 0 to 15.
Examples	**`CONTrol:AUXiliary:C:DATA 15`** **`For Positive Logic Port C lines C0, C1, C2, C3 go High or if in Negative Logic they go Low.`** **`CONTrol:AUXiliary:C:DATA?`** A returned value of 15 when in Positive Logic indicates Port C lines C0, C1, C2, C3 are High, or if in Negative Logic they are Low.
Query Syntax	CONTrol:AUXiliary:C:DATA?
Return Type	Integer
Overlapped?	No
Default	0

CONTrol:AUXiliary:C:LOGic <char>

(Read-Write) Reads and writes the logic mode of Port C on the AUX IO. This port is connected to Port C of the Handler IO connector. Therefore, it will have the same logic setting.

Parameters

<char>	Logic of Port C. Choose from: **POSitive** - when a value of one is written the associate line goes High **NEGative** - when a value of one is written the associate line goes Low When Port C is in Output/Write mode, a change in logic causes the output lines to change state immediately. For example, Low levels change to High levels. When Port C is in Input/Read mode, a change in logic does NOT cause the lines to change, but data read from Port C will reflect the change in logic.
Examples	CONT:AUX:C:LOG POS 'Positive logic is applied to Port C data.
Query Syntax	CONTrol:AUXiliary:C:LOGic?
Return Type	Character
Overlapped?	No
Default	NEGative

CONTrol:AUXiliary:C:MODE <char>

(Read-Write) Sets Port C to read or write mode. This port is connected to Port C of the Handler IO connector. Therefore, it will have the same mode setting.

NOTE: When Port C is set to INPut mode, data writes are NOT applied to the lines. MODE must be set to OUTPut mode before writing.

Parameters

<char>	INPut - set the port for reading OUTPut - set the port for writing
Examples	`CONT:AUX:C:MOD INP 'set Port C to Input Mode for reading.` CONTrol:AUXiliary:C:MODE? 'queries the input/output mode that the port set to.
Query Syntax	CONTrol:AUXiliary:C:MODE?
Return Type	Character
Overlapped?	No
Default	INPut

CONTrol:AUXiliary:FOOTswitch?

(Read) Reads the Auxiliary connector Footswitch Input (pin 20 of the AUX IO connector).

Examples	CONT:AUX:FOOT? control:auxiliary:footswitch?
Return Type	Boolean **True (or 1)** = pressed **False (or 0**) = released

Overlapped? No
Default False (0) - Released

CONTrol:AUXiliary:FOOTswitch:MODe <IGNore| SWEep| RECall |MACRo>

(Read-Write) This command sets the mode of the "FootSwitch In" line on the Auxiliary IO. These mode settings determine what occurs when the footswitch is pressed. Examine your results carefully when using these command modes. Refer to each mode description and associated notes, also see the FootSwitch In pin description in the Auxiliary IO connector documentation.

Parameters

IGNore - While in this mode any Footswitch presses are ignored.

SWEep - While in this mode a Footswitch press will trigger the sweep.

NOTE: The instrument must be in Manual Trigger Mode.

RECall - While in this mode a Footswitch press will recall an instrument state. When more than one state are available each footswitch press recalls the next state, then starts over from the beginning.

NOTE: It's possible to override the current mode with a recalled state. For instance when the current footswitch mode is RECall and the footswitch mode in the recalled state is IGNore the mode will change to IGNore. If this occurs additional footswitch presses will be ignored.

MACRo - While in this mode a Footswitch press will load and run a macro. When more than one macro are available each successive footswitch press loads and runs the next macro, then starts over from the beginning.

NOTE: It's possible to override the current mode when using the MACRo mode. For instance with the current footswitch mode set to MACRo and then running a macro containing a Preset the Preset will change the mode to IGNore because the default-preset mode is IGNore. If this occurs additional footswitch presses will be ignored.

Examples

`CONT:AUX:FOOT:MODe MACRo This sets the footswitch mode to MACRo causing a macro to be loaded and run with a footswitch press.`

CONTrol:AUXiliary:FOOTswitch:MODe? This query returns the footswitch mode setting.

Query Syntax CONTrol:AUXiliary:FOOTswitch:MODE?
Return Type Character IGNore or SWEep or RECall or MACRo

Overlapped? No
Default IGNore

CONTrol:AUXiliary:INPut:VOLTage?

(Read-Only) Reads the ADC input voltage from pin 14 of the AUX IO connector.

Examples CONT:AUX:INPut:VOLT?
control:auxiliary:input:voltage?

Return Type REAL
Overlapped? No
Default Not Applicable

CONTrol:AUXiliary:OUTPut[1|2]:MODe <WAIT|NOWait>

(Read-Write) This command sets the mode of the selected "Analog Out" line on the Auxiliary IO. The modes give the user the option to have the requested voltage applied immediately or not until the sweep is done. Also see the description for "Analog Out 1, 2" in the Auxilliary IO connector documentation.

Parameters

	WAIT - While in this mode any voltage changes sent to the selected analog out will only get applied to the output between sweeps.
	NOWait - While in this mode any voltage changes sent to the selected analog out will occur right away without waiting until the end of a sweep.
Examples	**`CONT:AUX:OUTP1:MOD WAIT This sets the mode so that voltages sent to "Analog Out 1" are only applied at the end of a sweep.`**
	CONT:AUX:OUTP2:MOD? This query returns the current mode for "Analog Out 2".
Query Syntax	CONTrol:AUXiliary:OUTPut2:MODe?
	’Reads the output mode
Return Type	Char WAIT or NOWait
Overlapped?	No
Default	WAIT

CONTrol:AUXiliary:OUTPut<out>:VOLTage <num>

(Read-Write) Sets and reads voltages on the DAC/Analog Output 1 and Output 2 (pins 2 and 3) of the Auxiliary IO connector.

Parameters

<out>	DAC output number. Choose from:
	1 - DAC Output 1 (pin 2)
	2 - DAC Output 2 (pin 3)
<num>	Output Voltage. Choose a voltage value between **-10** and **+10** volts
Examples	CONT:AUX:OUTP1:VOLT 5 control:auxiliary:output2:voltage 5
Query Syntax	CONTrol:AUXiliary:OUTPut<out>:VOLTage?
	’Reads the output DAC voltage
Return Type	REAL
Overlapped?	No
Default	0

CONTrol:AUXiliary:PASSfail:LOGic <char>

(Read-Write) Sets the logic of the PassFail line (pin 12) on the AUX IO connector. This line is connected internally to the PassFail line of the Material Handler IO (pin 33).

Parameters

<char>	Choose from: **POSitive** - Causes the PassFail line to have positive logic (high = pass, low = fail).
	NEGative - Causes the PassFail line to have negative logic (high = fail, low = pass).

Examples	CONT:AUX:PASS:LOG POS control:auxiliary:passfail:logic negative
Query Syntax **Return Type**	CONTrol:AUXiliary:PASSfail:LOGic? Character
Overlapped? **Default**	No POSitive

CONTrol:AUXiliary:PASSfail:MODe <char>

(Read-Write) Sets and reads the mode for the PassFail line (pin 12) on the AUX IO connector. This line is hardwired to the PassFail line (pin 33) of the Material Handler IO connector.

Parameters

<char>	Choose from: **PASS** - the line stays in PASS state. When a device fails, then the line goes to FAIL state after the SweepEnd line is asserted. **FAIL** - the line stays in FAIL state. When a device passes, then the line goes to PASS state after the SweepEnd line is asserted. **N**OWait - the line stays in PASS state. When a device fails, then the line goes to FAIL state IMMEDIATELY.
Examples	CONT:AUX:PASS:MODE NOW control:auxiliary:passfail:mode fail
Query Syntax **Return Type**	CONTrol:AUXiliary:PASSfail:MODE? Character
Overlapped? **Default**	No NOWait

CONTrol:AUXiliary:PASSfail:SCOPe <char>

(Read-Write) Sets and reads the scope of the PassFail line on the AUX IO connector. This line is connected to the PassFail line of the Handler IO connector. Therefore, it will have the same scope.

Parameters

<char>	Choose from: **Channel** - The PassFail line returns to its default state before sweeps on the next channel start. (A channel measurement may require several sweeps.) **Global** - The PassFail line returns to its default state before the sweeps for the next **triggerable** channel start. The default state of the passFail line (before a measurement occurs and after a failure occurs) is set by CONTrol:AUXiliary:PASSfail:MODe
Examples	CONT:AUX:PASS:SCOP CHAN control:auxiliary:passfail:scope sweep
Query Syntax **Return Type**	CONTrol:AUXiliary:PASSfail:SCOPe? Character
Overlapped? **Default**	No CYCLe

CONTrol:AUXiliary:SWEepend <char>

(Read-Write) Specifies the event that will cause the AUX IO Sweep End line (pin 11) to go to a low (false) state. The line will return to a high state after the appropriate calculations are complete. This line is connected internally to the Sweep End line of the Material Handler IO.

Parameters

<char>	Choose from: **Sweep -** the line goes low when each sweep is complete. **Channel** - The line goes low when all of the sweeps for each channel is complete. **Global** - The line goes low when all the sweeps for all channels are complete. The default state of the passFail line (before a measurement occurs and after a failure occurs) is set by CONTrol:AUXiliary:PASSfail:MODe.
Examples	CONT:AUX:SWE SWE control:auxiliary:sweepend channel
Query Syntax	CONTrol:AUXiliary:SWEepend?
Return Type	Character
Overlapped?	No
Default	SWEep

CONTrol:EXTernal:TESTset:DATa <addr>,<data>

(Read-Write) Reads and writes 13 bits of data to the specified address using the AD0 through AD12 lines of the external test set connector. The instrument generates the appropriate timing signals (strobes the address, then the data) to control an external test set.

Parameters

<addr>	Decimal equivalent of the 13 bit binary address.
<data>	Decimal equivalent of the 13 bit binary data
Examples	CONT:EXT:TEST:DATA 12,3 CONTrol:external:testset:data 12,3
Query Syntax	CONTrol:EXTernal:TESTset:DATA? <addr> 'Reads the decimal equivalent of the binary data from the specified address
Return Type	Integer
Overlapped?	No
Default	Not Applicable

CONTrol:EXTernal:TESTset:INTerrupt?

(Read-Only) Reads the boolean state of the Interrupt In line (pin 13) on the external test set connector.

Examples	CONT:EXT:TEST:INT? control:external:testset:interrupt?
Return Type	Boolean False (0) - the line is being held at a TTL High. True (1) - the line is being held at a TTL Low.
Overlapped?	No
Default	Not Applicable

CONTrol:EXTernal:TESTset:RAWData <data>

(Read-Write) Reads and writes 16 bits of data through the AD0 through AD12 and three timing lines of the external test set connector. Does NOT generate appropriate timing signals.

Use of this command requires detailed knowledge of all 16 bits. Refer to the Data format table.

Note: During a WRITE, Bit 13 must always be low. Otherwise Bit 0-12 will tristate

Parameters

<data>	Decimal equivalent of the binary data. Format of data **WRITTEN** with RAWData:

Pin	Bit	Signal name
22	0	AD0*
23	1	AD1*
11	2	AD2*
10	3	AD3*
9	4	AD4*
21	5	AD5*
20	6	AD6*
19	7	AD7*
6	8	AD8*
5	9	AD9*
4	10	AD10*
17	11	AD11*
3	12	AD12*
25	13	RLW
24	14	LDS
8	15	LAS

* This Output will float if RLW (bit-13) is set high

Examples

CONT:EXT:TEST:RAWD 8001
CONTrol:external:testset:rawdata 1234

Query Syntax CONTrol:EXTernal:TESTset:RAWData?
Return Format Format of data **READ** with RAWData?

Pin	Bit	Signal name
22	0	AD0*
23	1	AD1*
11	2	AD2*
10	3	AD3*
9	4	AD4*
21	5	AD5*
20	6	AD6*
19	7	AD7*
6	8	AD8*

5	9	AD9*
4	10	AD10*
17	11	AD11*
3	12	AD12*
2	13	Sweep Holdoff In
13	14	Interrupt In (inverted internally)
na	15	Always Zero, grounded internally

*These lines are dependent on the state of RLW (pin25).
Writing a 0(low) to RLW will set lines AD0-AD12 to write mode.
Writing a 1(high) to RLW will set lines AD0-AD12 to read mode.

Return Type Integer

Overlapped? No
Default Not Applicable

CONTrol:EXTernal:TESTset:SWEepholdoff?

(Read-Only) Reads the Sweep Holdoff line (pin 2) on the external test set connector.

Examples CONT:EXT:TEST:SWE?
control:external:testset:sweepholdoff?

Return Type Boolean

TRUE (1) - the pin is set to a TTL High
FALSE (0) - the pin is set to TTL Low

Overlapped? No
Default Not Applicable

CONTrol:HANDler:C:MODE <char>

(Read-Write) Sets and reads the direction of data flow for Port C.

Parameters

<char> Direction of flow. Choose from:

INPut - Port C is used to input data
OUTPut - Port C is used to output data

Examples CONT:HAND:C:MODE INP
control:handler:c:mode output

Query Syntax CONTrol:HANDler:C:MODE?
Return Type Character

Overlapped? No
Default INPut

CONTrol:HANDler:D:MODE <char>

(Read-Write) Sets and reads the direction of data flow for Port D

Parameters

<char> Direction of flow. Choose from:

INPut - Port D is used to input data
OUTPut - Port D is used to output data

Examples CONT:HAND:D:MODE INP
control:handler:d:mode output

Query Syntax CONTrol:HANDler:D:MODE?
Return Type Character

Overlapped? No
Default Input

CONTrol:HANDler:<port>[:DATa] <num>

(Read-Write) Writes and reads data on the specified port.

Parameters

<port> Port identifier to set bits for. Choose from:

A,B,C,D,E,F,G,H

<num> The number of the data bits to set. Refer to the following table for the maximum number for each port. The minimum number for each port is 0.

Port	Max allowable <num>	MSB...LSB 23..0	
A	255	A7...A0	Write-only
B	255	B7...B0	Write-only
C	15	C3...C0	Read-Write
D	15	D3...D0	Read-Write
E	255	D3...D0 + C3...C0	Read-Write
F	65535	B7...B0 + A7...A0	Write-only
G	1048575	C3...C0 + B7...B0 + A7...A0	Write-only
H	16777215	D3...D0 + C3...C0 + B7...B0 + A7...A0	Write-only

Note: When writing to port G, port C must be set to output mode
When writing to port H, both port C and port D must be set to output mode. Use CONT:HAND:C:MODE OUTP and CONT:HAND:D:MODE OUTP

Examples CONT:HAND:A 254
control:handler:c:data 12

Query Syntax CONTrol:HANDler:<port>:DATA?
Return Type Integer

Overlapped? No
Default Not Applicable

CONTrol:HANDler:INPut?

(Read-Only) Reads a hardware latch that captures low to high transition on Input1 of the Material Handler IO. Reading the latch causes it to reset and is ready for the next transition. The hardware latch is only capable of capturing one transition per query. Additional transitions are ignored until after the next query.

Examples CONT:HAND:INP?

control:handler:input?

Return Type Integer - Returns a value of zero or one.

One - A low to high transition occurred at Input1 since the last time it was queried.

Zero - No low to high transition occurred. After the query the latch is reset and is ready for the next input. If no low to high transitions occur additional queries will return zero.

Momentarily grounding or driving Input1 low, then high, will cause a transition to be detected and latched.

Overlapped? No

Default 0

CONTrol:HANDler:LOGic <char>

(Read-Write) Sets the logic of the Data ports A-H on the Handler connector. Some of these lines are connected internally to the AuxIO.

Parameters

<char> Choose from:

POSitive- Causes the Port lines to have positive logic (high = 1, low = 0).

NEGative- Causes the Port lines to have negative logic (high = 0, low = 1).

For ports that are in output (write) mode, a change in logic causes the output lines to change state immediately. For example, Low levels change immediately to High levels.

For ports that are in input (read) mode (C,D,E only), a change in logic will be reflected when data is read from that port. For example, if a line read 0, the next read after a logic change will read 1.

Examples CONT:HAND:LOG POS
control:handler:logic negative

Query Syntax CONTrol:HANDler:LOGic?

Return Type Character

Overlapped? No

Default POSitive

CONTrol:HANDler:OUTPut<num>[:DATa] <num2>

(Read-Write) Sets or reads whether the specified output line is High or Low.

Parameters

<num> Output port. Choose from:

1 - output 1(default)

2 - output 2

<num2> **0 - Low**

1 - High

Examples CONT:HAND:OUTPut1 1
control:handler:output2:data 0

Query Syntax CONTrol:HANDler:OUTPut<num>:DATA?

Return Type	Integer (0 or 1)
Overlapped?	No
Default	0 - Low

CONTrol:HANDler:OUTPut<num>:USER[:DATa] <num2>

(Read-Write) Sets or reads whether the specified USER output line is High or Low.

Parameters

<num>	USER Output port. Choose from: **1 -** User output 1(default) **2 -** User output port.
<num2>	**0 - Low** **1 - High**
Examples	CONT:HAND:OUTPut1:USER 1 control:handler:output2:user:data 0
Query Syntax	CONTrol:HANDler:OUTPut<num>:USER:DATA?
Return Type	Integer (0 or 1)
Overlapped?	No
Default	0 - Low

CONTrol:HANDler:PASSfail:LOGic <char>

(Read-Write) Sets the logic of the PassFail line of the Material Handler IO (pin 33). This line is connected internally to the PassFail line (pin 12) on the AUX IO connector.

Parameters

<char>	Choose from: **POSitive**- Causes the PassFail line to have positive logic (high = pass, low = fail). **NEGative**- Causes the PassFail line to have negative logic (high = fail, low = pass).
Examples	CONT:HAND:PASS:LOG POS control:handler:passfail:logic negative
Query Syntax	CONTrol:HANDler:PASSfail:LOGic?
Return Type	Character
Overlapped?	No
Default	POSitive

CONTrol:HANDler:PASSfail:MODe <char>

(Read-Write) Sets the mode for the PassFail line (pin 33) of the Material Handler IO connector. This line is hardwired to the PassFail line (pin 12) on the AUX IO connector.

Parameters

<char>	Choose from: **PASS**- the line stays in PASS state. When a device fails, then the line goes to fail after the Sweep End line is asserted. **FAIL**- the line stays in FAIL state. When a device passes, then the line goes to PASS state after the Sweep End line is asserted. **N**OWait- the line stays in PASS state. When a device fails, then the line

goes to fail IMMEDIATELY.

Examples CONT:HAND:PASS:MODE NOW
control:handler:passfail:mode fail

Query Syntax CONTrol:HANDler:PASSfail:MODE?
Return Type Character

Overlapped? No
Default NOWait

CONTrol:HANDler:PASSfail:SCOPe <char>

(Read-Write) Sets and reads scope mode of the PassFail line on the HANDLER IO. This line is connected to the PassFail line of the Handler IO connector. Therefore, it will have the same scope.

Parameters

<char> Choose from:

CHANnel- The PassFail line returns to its default state before sweeps on the next channel start. (A channel measurement may require several sweeps.)

GLOBal - The PassFail line returns to its default state before the sweeps for the next triggerable channel start.

The default state of the passFail line (before a measurement occurs) and after a failure occurs is set by CONTrol:HANDler:PASSfail:MODe

Examples CONT:HAND:PASS:SCOP CHAN
control:handler:passfail:scope sweep

Query Syntax CONTrol:HANDler:PASSfail:SCOPe?
Return Type Character

Overlapped? No
Default GLOBal

CONTrol:HANDler:SWEepend <char>

(Read-Write) Specifies the event that will cause the Handler Sweep End line to go to a low (false) state. The line will return to a high state after the appropriate calculations are complete. This line is connected internally to the Sweep End line of the AUX IO connector.

Parameters

<char> Choose from:

SWEep - the line goes low when **each sweep** is complete

CHANnel - the line goes low when **all the sweeps for each channel** is complete.

GLOBal - the line goes low when **all sweeps for all channels** are complete.

The default state of the passFail line (before a measurement occurs) and after a failure occurs is set by CONTrol:HANDler:PASSfail:MODe

Examples CONT:HAND:SWE SWE
control:handler:sweepend channel

Query Syntax CONTrol:HANDler:SWEepend?
Return Type Character

Overlapped?	No
Default	SWEep

CONTrol:SIGNal <conn>,<char>

(Read-Write) Enables external edge triggering in the PNA. To receive trigger signals from an external source, the PNA must be in External trigger mode. Edge triggering is only available on PNA models E8361A, E8362B, E8363B, and E8364B. For more information, see Edge Triggering.

Parameters

<conn> Rear Panel connector to send or receive trigger signals. Choose from:

BNC1 Trigger IN from external source (Trigger IN BNC connector)

AUXT Trigger IN from external source (AUX IO connector Pin 19)

Note: Only one of the input connectors is active at a time. When a command is sent to one, the PNA automatically makes the other INACTIVE.

BNC2 Trigger OUT (Trigger OUT BNC connector). Only useful in point sweep mode.

<char> When <conn> is set to either **BNC1** or **AUXT** choose from:

TIENEGATIVE - (Trigger In Edge Negative) - Triggers the PNA when receiving a negative going signal

TIEPOSITIVE - (Trigger In Edge Positive) - Triggers the PNA when receiving a positive going signal

TILLOW - (Trigger In Level Low) - Triggers the PNA when receiving a low level signal

TILHIGH - (Trigger In Level High) - Triggers the PNA when receiving a High-level signal

INACTIVE - Disables the specified connector.

Note: The channel to be triggered must be in point sweep mode

When <conn> is set to **BNC2** choose from:

TOPPAFTER - (Trigger Out Pulse Positive After) - Sends a POSITIVE going TTL pulse at the END of each point during the sweep.

TOPBEFORE - (Trigger Out Pulse Positive Before) - Sends a POSITIVE going TTL pulse at the START of each point during the sweep.

TOPNAFTER - (Trigger Out Pulse Negative After) - Sends a NEGATIVE going TTL pulse at the END of each point during the sweep.

TOPNBEFORE - (Trigger Out Pulse Negative Before) - Sends a NEGATIVE going TTL pulse at the START of each point during the sweep.

INACTIVE - Disables the specified connector.

Examples

```
CONT:SIGN BNC1,TIENEGATIVE
 control:signal bnc2,topbefore
```

Query Syntax CONTrol:SIGNal <conn>?

In addition to the arguments listed above, the following is also a possible returned value:

NAVAILABLE - This feature is not available on this PNA

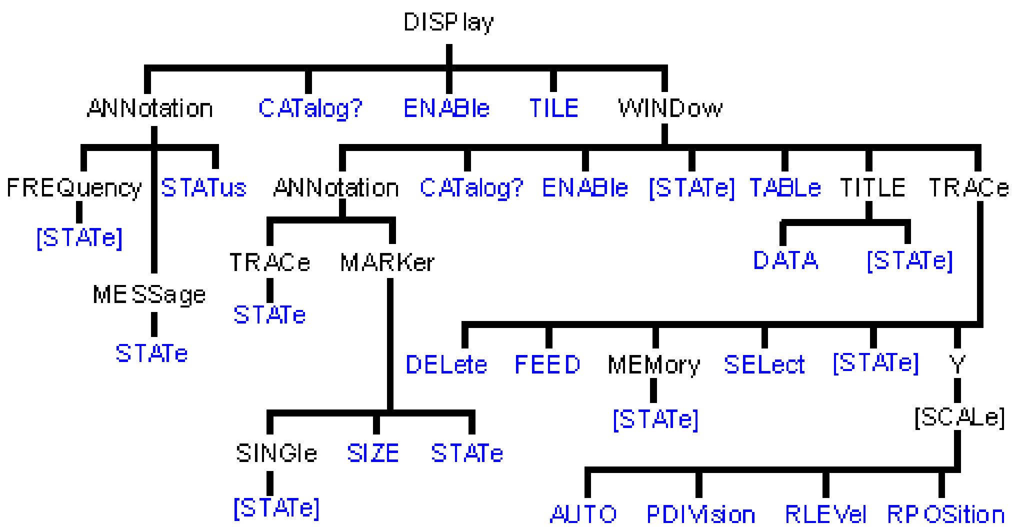
Return Type Character

Overlapped? No

Default BNC1 = INACTIVE
BNC2 = INACTIVE
AUXT = TILHIGH

Display Commands

Controls the settings of the front panel screen.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands
- Learn about Screen Setup

DISPlay:ANNotation:FREQuency[:STATe] <ON | OFF>

(Read-Write) Turns frequency information on the display title bar ON or OFF for all windows.

Parameters

<ON | OFF> **ON** (or 1) - turns frequency annotation ON.
OFF (or 0) - turns frequency annotation OFF.

Examples DISP:ANN:FREQ ON
display:annotation:frequency:state off

Query Syntax DISPlay:ANNotation:FREQuency[:STATe]?

Return Type Boolean (1 = ON, 0 = OFF)

Overlapped?	No
Default	**ON (1)**

DISPlay:ANNotation:MESSage:STATe <ON | OFF>

(Read-Write) Enables and disables error pop-up messages on the display.

Parameters

<ON | OFF>	**ON** (or 1) - enables error pop-up messages **OFF** (or 0) - disables error pop-up messages
Examples	DISP:ANN:MESS:STAT ON display:annotation:message:state off
Query Syntax	DISPlay:ANNotation:MESSage:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	**ON (1)**

DISPlay:ANNotation:STATus <ON|OFF>

(Read-Write) Turns the status bar at the bottom of the screen ON or OFF. The status bar displays information for the active window.

Parameters

<ON | OFF>	**ON** (or 1) - turns status bar ON. **OFF** (or 0) - turns status bar OFF.
Examples	DISP:ANN:STAT ON display:annotation:status off
Query Syntax	DISPlay:ANNotation:STATus?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	**Last state that was set**

DISPlay:CATalog?

(Read-only) Returns the existing Window numbers.

Return Type	String of Character values, separated by commas
Example	**Two windows with numbers 1 and 2 returns:** "1,2"
Overlapped?	No
Default	Not applicable

DISPlay:ENABLE <ON | OFF>

(Read-Write) Specifies whether to disable or enable all analyzer display information **in all windows** in the analyzer application. Marker data is not updated. More CPU time is spent making measurements instead of updating the display.

Parameters

<ON | OFF>	**ON** (or 1) - turns the display ON. **OFF** (or 0) - turns the display OFF.
Examples	DISP:ENAB ON display:enable off

Query Syntax	DISPlay:ENABle?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

DISPlay[:TILE]

(Write-only) Tiles the windows on the screen.

Examples	DISP display:tile
Overlapped?	No
Default	Not Applicable

DISPlay:WINDow<wnum>:ANNotation:MARKer:SINGle[:STATe] <bool>

(Read-Write) Either shows marker readout of only the active trace or all of the traces simultaneously. See other SCPI Marker commands

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<bool>	**ON** (or 1) - show a single marker per trace **OFF** (or 0) - show up to 4 markers per active trace
Examples	DISP:WIND:ANN:MARK:SING ON display:window:annotation:marker:single off
Query Syntax	DISPlay:WINDow:ANNotation:MARKer:SINGle?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

DISPlay:WINDow<wnum>:ANNotation:MARKer:SIZE <char>

(Read-Write) Specifies the size of the marker readout text. See other SCPI Marker commands

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<char>	Readout text size. Choose from:**NORMal I LARGe**
Examples	DISP:WIND:ANN:MARK:SIZE LARG display:window:annotation:marker:size normal
Query Syntax	DISPlay:WINDow:ANNotation:MARKer:SIZE?
Return Type	Character
Overlapped?	No
Default	NORMal

DISPlay:WINDow<wnum>:ANNotation:MARKer:STATe <ON|OFF>

(Read-Write) Specifies whether to show or hide the Marker data (when markers are ON) on the selected window. See other SCPI Marker commands

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<ON I OFF>	**ON** (or 1) - turns marker data ON. **OFF** (or 0) - turns marker data OFF.
Examples	DISP:WIND:ANN:MARK:STAT ON

display:window:annotation:marker:state off

Query Syntax DISPlay:WINDow:ANNotation:MARKer:STATe?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default ON

DISPlay:WINDow<wnum>:ANNotation:TRACe:STATe <ON|OFF>

(Read-Write) Specifies whether to show or hide the Trace Status buttons on the left of the display.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<ON | OFF>	**ON** (or 1) - turns the buttons ON. **OFF** (or 0) - turns the buttons OFF.

Examples DISP:WIND:ANN:TRAC:STAT ON
display:window:annotation:trace:state off

Query Syntax DISPlay:WINDow:ANNotation:TRACe:STATe?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default ON

DISPlay:WINDow<wnum>:CATalog?

(Read-only) Returns the trace numbers for the specified window.

Parameters

<wnum> Any existing window number (**1** to **4**); if unspecified, value is set to 1.

Return Type String of Character values, separated by commas
Example **Window 1 with four traces:**
DISPlay:WINDow1:CATalog?
Returns:
"1,2,3,4"

Overlapped? No
Default Not applicable

DISPlay:WINDow<wnum>:ENABle <ON | OFF>

(Read-Write) Specifies whether to disable or enable all analyzer display information **in the specified window**. Marker data is not updated. More CPU time is spent making measurements instead of updating the display.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<ON | OFF>	**ON** (or 1) - turns the display ON. **OFF** (or 0) - turns the display OFF.

Examples DISP:WIND:ENABle ON
display:window1:enable off

Query Syntax DISPlay:WINDow<wnum>:ENABle?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default ON

DISPlay:WINDow<wnum>[:STATe] <ON | OFF>

Write to create or delete a window on the screen or Read whether a window is present.

Parameters

<wnum>	Window number to create; choose any integer between: **1** and **4**
<ON | OFF>	**ON** (or 1) - The window <wnum> is created. **OFF** (or 0) - The window <wnum> is deleted.
Examples	DISP:WIND ON display:window2:state off
Query Syntax	DISPlay:WINDow<wnum>[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	Window number "1" **ON**

DISPlay:WINDow<wnum>:TABLe <char>

Write to show the specified table at the bottom of the analyzer screen or Read to determine what table is visible.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1
<char>	Table to show. Choose from: **OFF | MARKer | LIMit | SEGMent**
Examples	DISP:WIND:TABLe SEGM display:window:table off
Query Syntax	DISPlay:WINDow:TABLe?
Overlapped?	No
Default	OFF

DISPlay:WINDow<wnum>:TITLe:DATA <string>

(Read-Write) Sets data in the window title area. The title is turned ON and OFF with DISP:WIND:TITL:STAT OFF.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<string>	Title to be displayed. Any characters, enclosed with quotes. If the title string exceeds 50 characters, an error will be generated and the title not accepted. Newer entries replace (not append) older entries.
Examples	DISP:WIND:TITL:DATA 'hello' display:window2:title:data 'hello'
Query Syntax	DISPlay:WINDow<wnum>:TITLe:DATA?
Return Type	String
Overlapped?	No
Default	NA

DISPlay:WINDow<wnum>:TITLe[:STATe] <ON | OFF>

(Read-Write) Turns display of the title string ON or OFF. When OFF, the string remains, ready to be redisplayed when turned back ON.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns the title string ON. **OFF** (or 0) - turns the title string OFF.
Examples	DISP:WIND:TITL ON Display:window1:title:state off
Query Syntax	DISPlay:WINDow<wnum>:TITLe[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

DISPlay:WINDow<wnum>:TRACe<tnum>:DELete

(Write-only) Deletes the specified trace from the specified window. The measurement parameter associated with the trace is not deleted.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	The number of the trace to be deleted; if unspecified, value is set to 1
Examples	DISP:WIND:TRAC:DEL display:window2:trace2:delete
Query Syntax	Not applicable
Overlapped?	No
Default	NA

DISPlay:WINDow<wnum>:TRACe<tnum>:FEED <name>

(Write-only) Creates a new trace <tnum> and associates (feeds) a measurement <name> to the specified window<wnum>. This command should be executed immediately after creating a new measurement with CALC:PAR:DEF<name>,<parameter>.

To feed the same measurement to multiple traces, create another measurement with the same <name>,<parameter> using the CALC:PAR:DEF command. The analyzer will collect the data only once.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Trace number to be created. Choose any Integer between: **1** and **4**
<name>	Name of the measurement that was defined with CALC:PAR:DEF**<name>**,<parameter>
Examples	DISP:WIND:TRAC:FEED 'test' display:window2:trace2:feed 'test'
Query Syntax	Not applicable
Overlapped?	No
Default	"CH1_S11"

DISPlay:WINDow<wnum>:TRACe<tnum>MEMory[:STATe] <ON | OFF>

(Read-Write) Turns the memory trace ON or OFF.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.

<tnum>	Any existing trace number; if unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns the memory trace ON. **OFF** (or 0) - turns the memory trace OFF.

Examples	DISP:WIND:TRAC:MEM ON display:window2:trace2:memory:state off
Query Syntax	DISPlay:WIND<wnum>:TRACe<tnum>:MEMory[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

DISPlay:WINDow<wnum>:TRACe<tnum>:SELect

(Write-only) Activates the specified trace in the specified window for front panel use.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1

Examples	DISP:WIND:TRAC:SEL display:window2:trace2:select
Query Syntax	Not applicable
Overlapped?	No
Default	NA

DISPlay:WINDow<wnum>:TRACe<tnum>[:STATe] <ON | OFF>

(Read-Write) Turns the display of the specified trace in the specified window ON or OFF. When OFF, the measurement behind the trace is still active.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns the trace ON. **OFF** (or 0) - turns the trace OFF.

Examples	DISP:WIND:TRAC ON display:window2:trace2:state off
Query Syntax	DISPlay:WIND<wnum>:TRACe<tnum>[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:AUTO

(Write-only) Performs an **Autoscale** on the specified trace in the specified window, providing the best fit display. Autoscale is performed only when the command is sent; it does NOT keep the trace autoscaled indefinitely.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1

Examples	DISP:WIND:TRAC:Y:AUTO display:window2:trace2:y:scale:auto

Query Syntax Not applicable

Overlapped? No
Default Not applicable

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:PDIVision <num>

(Read-Write) Sets the Y axis **Per Division** value of the specified trace in the specified window.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1
<num>	Units / division value. The range of acceptable values is dependent on format and domain. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples
DISP:WIND:TRAC:Y:PDIV 1
display:window2:trace2:y:scale:pdivision maximum

Query Syntax DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:PDIVision?
Return Type Character

Overlapped? No
Default 10

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RLEVel <num>

(Read-Write) Sets the Y axis Reference Level of the specified trace in the specified window.

Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1
<num>	Reference level value. The range of acceptable values is dependent on format and domain. **Note:** This command will accept MIN or MAX instead of a numeric parameter. See SCPI Syntax for more information.

Examples
DISP:WIND:TRAC:Y:RLEV 0
display:window2:trace2:y:scale:rlevel minimum

Query Syntax DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RLEVel?
Return Type Character

Overlapped? No
Default NA

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RPOSition <num>

(Read-Write) Sets the **Reference Position** of the specified trace in the specified window

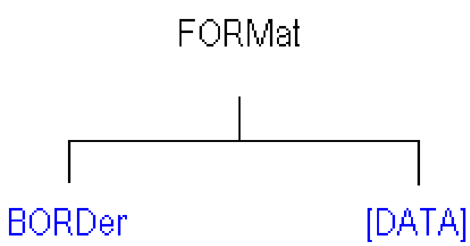
Parameters

<wnum>	Any existing window number (**1** to **4**); if unspecified, value is set to 1.
<tnum>	Any existing trace number; if unspecified, value is set to 1
<num>	Reference position on the screen measured in horizontal graticules from the bottom. The range of acceptable values is dependent on format and domain. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples	DISP:WIND:TRAC:Y:RPOS 0 display:window2:trace2:y:rposition maximum
Query Syntax	DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RPOSition?
Return Type	Character
Overlapped?	No
Default	5

Format Commands

Specifies the way that data will be transferred when moving large amounts of data. These commands will affect data that is transferred with the CALC:DATA and CALC:RDATA commands.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.

FORMat:BORDer <char>

(Read-Write) Set the byte order used for GPIB data transfer. Some computers read data from the analyzer in the reverse order. This command is only implemented if FORMAT:DATA is set to :REAL. If FORMAT:DATA is set to :ASCII, the swapped command is ignored.

Parameters

<char>	Choose from: **NORMal -** Use when your controller is anything other than an IBM compatible computers **SWAPped** - for IBM compatible computers
Examples	FORM:BORD SWAP format:border normal
Query Syntax	FORMat:BORDer?
Overlapped?	No
Default	Normal

FORMat[:DATA] <char>

(Read-Write) Sets the data format for data transfers. To transfer measurement data, use the CALC:DATA command.

To transfer Source Power correction data, use
SOURce:POWer:CORRection:COLLect:TABLe:DATA
SOURce:POWer:CORRection:COLLect:TABLe:FREQuency
SOURce:POWer:CORRection:DATA

Parameters

<char>	Choose from: **REAL,32** - (default value for REAL) Best for transferring large amounts of measurement data. **REAL,64** - Slower but has more significant digits than REAL,32. Use REAL,64 if you have a computer that doesn't support REAL,32. **ASCii,0** - The easiest to implement, but very slow. Use if small amounts of data to transfer. For more information, see Transferring Measurement Data
Examples	FORM REAL,64 format:data ascii
Query Syntax	FORMat:DATA?
Return Type	Character,Character
Overlapped?	No
Default	REAL,32

Hardcopy Command

Learn about Printing

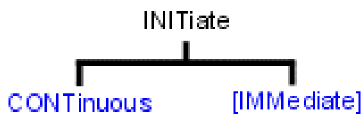
HCOPy[:IMMediate]

(Write-only) Prints the screen to the default printer.

Examples	HCOP hcopy:immediate
Query Syntax	Not applicable
Overlapped?	No
Default	Not Applicable

Initiate Commands

Controls triggering signals



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Triggering

INITiate:CONTinuous <boolean>

(Read-Write) Specifies whether the analyzer sends Continuous sweep triggers to triggerable channels or enables Manual triggering.

Parameters

<boolean>	**ON** (or 1) - Continuous sweep mode. **OFF** (or 0) - Manual sweep mode.
Examples	INIT:CONT ON initiate:continuous off
Query Syntax	INITiate:CONTinuous?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

INITiate<cnum>[:IMMediate]

(Write-only) Stops the current sweeps and immediately sends a trigger to the specified channel. (Same as Sweep \ Trigger \ Trigger!)

- If the specified channel is in HOLD, it will sweep one time and return to HOLD when complete.
- If Trigger:Scope = Global, all channels will receive a trigger.
- If Trigger:Scope = Channel (only the active channel receives a trigger) and the specified channel is not the active channel, the specified channel will NOT receive a trigger signal.
- If the specified channel is NOT in Manual trigger (INIT:CONT OFF), the analyzer will return an error.

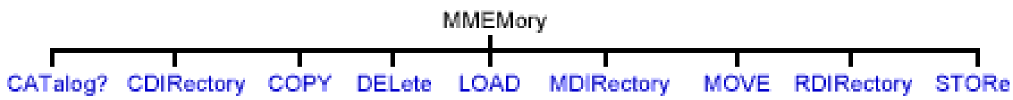
If channel <cnum> does not exist, the analyzer will return an error.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	INIT initiate2:immediate
Query Syntax	Not applicable
Overlapped?	Yes
Default	Not applicable

Memory Commands

The memory commands control saving and loading instrument states to the hard drive.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Save / Recall

All MMEM files have an extension according to their type.

1. Binary filetype:
   - .sta - Instrument State
   - .cal - Calibration file
   - .cst - Both Instrument State and Calibration file
2. ASCII filetype (MDIF or Touchstone formats):
   - .s1p
   - .s2p
   - .s3p

The default folder is "C:\Program Files\Agilent\Network Analyzer\Documents"

You can change the active directory using MMEMory:CDIRectory, or you can use an absolute path name to specify all MMEM files and folders.

MMEMory:CATalog[:<char>]? [<folder>]

(Read-only) Returns a comma-separated string of file names that are in the specified folder. If there are no files of the specified type, "NO CATALOG" is returned.

Parameters

<char>	The type of files to list. Choose from: **STATe** - Instrument states (.sta) **CORRection** - Calibration Data (.cal) **CSTate** - Instrument state and Calibration data (.cst) If unspecified then ALL file types (even unknown types) are listed.
<folder>	String - Any existing folder name. If unspecified 'C:\Program Files\Agilent\Network Analyzer\Documents' is used.
Examples	MMEM:CAT? 'lists all files from the current folder mmemory:catalog:correction? 'C:\Program Files\Agilent\Network Analyzer\Documents' 'lists .cal files from the specified folder
Overlapped?	No
Default	Not applicable

MMEMory:CDIRectory <folder>

(Read-Write) Changes the folder name.

Parameters

<folder>	Any drive and folder name that already exists.
	If the same level as "C:\Program Files\Agilent\Network Analyzer\Documents", then no punctuation is required
	`MMEM:CDIR Service`
	If the new folder is at a different level than "C:\Program Files\Agilent\Network Analyzer\Documents", use a slash (\) before the folder name and enclose in quotes.
	`mmemory:cdirectory '\automation' 'changes default directory up one level.`
	You can use an absolute path to specify the new folder.
	mmemory:cdirectory 'c:\automation\service'

Query Syntax	MMEMory:CDIRectory? 'Returns the current folder name
Return Type	String
Overlapped?	No
Default	'C:\Program Files\Agilent\Network Analyzer\Documents'

MMEMory:COPY <file1>,<file2>

(Write-only) Copies file1 to file2. Extensions must be specified.

Parameters

<file1>	String - Name of the file to be copied.
<file2>	String - Name of the file to be created from file1.
Examples	MMEM:COPY 'MyFile.cst','YourFile.cst'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:DELete <file>

(Write-only) Deletes file. Extensions must be specified.

Parameters

<file>	String - Name of the file to be deleted.
Examples	MMEM:DEL 'MyFile.cst'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:LOAD[:<char>] <file>

(Write-only) Loads the specified file.

Parameters

<char>	The type of file to load. Choose from: **STATe** - Instrument states (.sta) **CORRection** - Calibration Data (.cal) **CSTate** - Instrument state and Calibration data (.cst)

	If <char> is unspecified, the extension must be included in the filename.
	If an extension is specified in <file> that does not agree with <char> then no action is taken.
<file>	String - Name of the file to be loaded. The default folder is used if unspecified in the filename.
Examples	MMEM:LOAD 'MyFile.cst' mmemory:load:state 'MyInstState'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:MDIRectory <folder>

(Write-only) Makes a folder.

Parameters

<folder>	String - Name of the folder to make.
Examples	MMEM:MDIR 'MyFolder' mmemory:mdirectory 'c:\NewFolder'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:MOVE <file1>,<file2>

(Write-only) Renames <file1> to <file2>. File extensions must be specified.

Parameters

<file1>	String - Name of the file to be renamed.
<file2>	String - Name of the new file.
Examples	MMEM:MOVE 'MyFile.cst','YourFile.cst'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:RDIRectory <folder>

(Write-only) Removes the specified folder.

Parameters

<folder>	String - Name of the folder to remove.
Examples	MMEM:RDIR 'MyFolder'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

MMEMory:STORe[:<char>] <file>

(Write-only) Stores the specified file (.sta, .cal, .cst, .s1p, .s2p, and .s3p.).

The ASCII file types (.s1p, .s2p, and .s3p.) may be valid only if the proper calibration is enabled for the current active measurement.

Example:

MMEM:STOR "myfile.s2p" stores an s2p file successfully if 2-Port calibration is enabled.

For more information on filetypes (see: Save recall a file)

Parameters

<char>	The type of file to store. Choose from: **STATe** - Instrument states (.sta) **CORRection** - Calibration Data (.cal) **CSTate** - Instrument state and Calibration data (.cst) No <char> is specified for s1p, s2p and s3p If unspecified, then the extension must be included in the filename. If an extension is specified in <file> that does not agree with <char> then no action is taken.
<file>	String - Name of any valid file that is not already in existence.
Examples	MMEM:STOR:STAT 'myState' mmemory:store 'c:\bin\myState.sta'
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

Output Command

Learn about Power

OUTPut[:STATe] <ON | OFF>

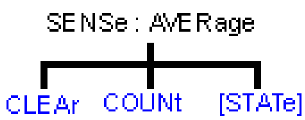
(Read-Write) Turns RF power from the source ON or OFF.

Parameters

<ON | OFF>	**ON** (or 1) - turns RF power ON. **OFF** (or 0) - turns RF power OFF.
Examples	OUTP ON output:state off
Query Syntax	OUTPut[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

Sens:Average Commands

Sets sweep-to-sweep averaging parameters. Averaging is a noise reduction technique that averages each data point over a user-specified number of sweeps. Averaging affects all of the measurements in the channel.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands.
- Learn about Averaging

SENSe<cnum>:AVERage:CLEar

(Write-only) Clears and restarts averaging of the measurement data. Must also set SENS:AVER[:STATe] ON

Parameters

<cnum>	Any existing channel number; if unspecified, value is set to 1.
Examples	SENS:AVER:CLE sense2:average:clear
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

SENSe<cnum>:AVERage:COUNt <num>

(Read-Write) Sets the number of measurement sweeps to combine for an average. Must also set SENS:AVER[:STATe] ON

Parameters

<cnum>	Any existing channel number; if unspecified, value is set to 1.
<num>	Number of measurement sweeps to average. Choose any number between: **1** and **1024**
Examples	SENS:AVER:COUN 999 sense2:average:count 73
Query Syntax	SENSe<cnum>:AVERage:COUNt?
Return Type	Character
Overlapped?	No
Default	1

SENSe<cnum>:AVERage[:STATe] <ON | OFF>

(Read-Write) Turns trace averaging ON or OFF.

Parameters

<cnum>	Any existing channel number; if unspecified, value is set to 1.
<ON | OFF>	**ON** (or 1) - turns averaging ON. **OFF** (or 0) - turns averaging OFF.
Examples	SENS:AVER ON sense2:average:state off
Query Syntax	SENSe<cnum>:AVERage[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	Off

Sense:Bandwidth Command

Learn about IF Bandwidth

SENSe<cnum>:BANDwidth | BWIDth[:RESolution] <num>

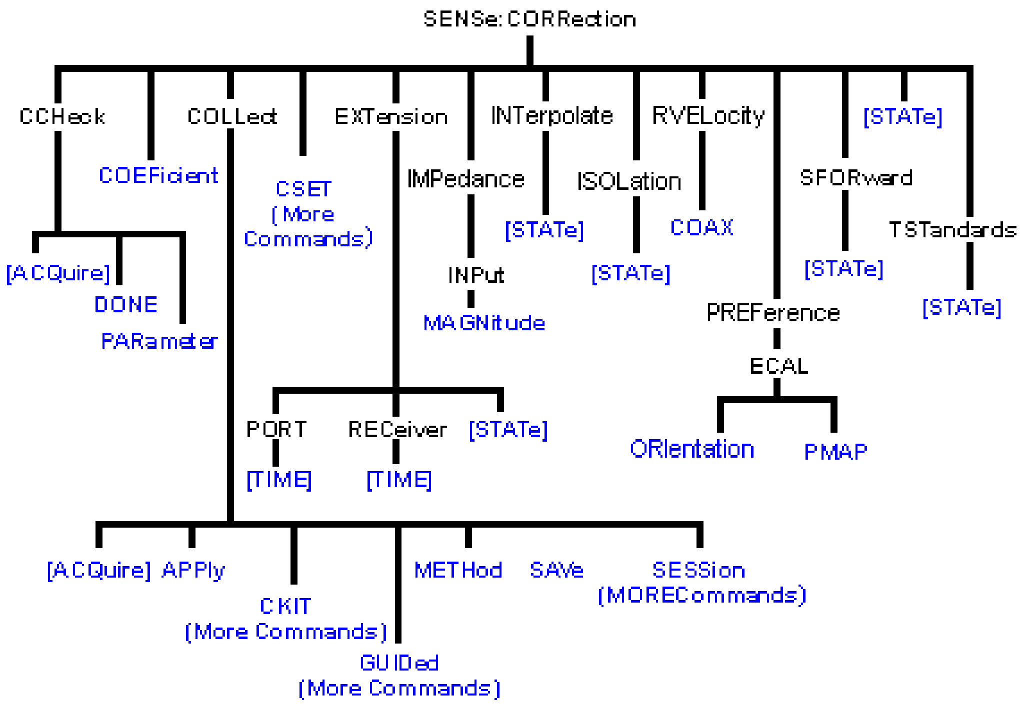
(Read-Write) Sets the bandwidth of the digital IF filter to be used in the measurement. The keywords BAND or BWID are interchangeable.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	IF Bandwidth in Hz. Choose from: **1 | 2 | 3 | 5 | 7 | 10 | 15 | 20 | 30 | 50 | 70 | 100 | 150 | 200 | 300 | 500 | 700 | 1k | 1.5k | 2k | 3k | 5k | 7k | 10k | 15k | 20k | 30k | 35k | 40k |** If a number other than these is entered, the analyzer will round up to the closest valid number (unless a number higher than the maximum in entered.)
Examples	SENS:BWID 1KHZ sense2:bandwidth:resolution 1000
Query Syntax	SENSe<cnum>:BANDwidth | BWIDth[:RESolution]?
Return Type	Character
Overlapped?	No
Default	35k

Sense:Correction Commands

Performs and applies measurement calibration and other error correction features.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands.
- Learn about Measurement Calibration

SENSe<cnum>:CORRection:CCHeck[:ACQuire] <mod>[,char]

(Write-only) Reads the 'confidence data' associated with the specified ECal module and puts it into memory. The measurement is selected using SENS:CORR:CCH:PAR. This command is compatible with *OPC.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1.
<mod>	ECAL Module that contains the confidence data. Choose from: • ECALA ECALB
[char]	Optional argument. Specifies which characterization within the ECal module that the confidence data will be read from. **CHAR0** Factory characterization (data that was stored in the ECal module by Agilent). Default if not specified. **CHAR1** User characterization #1

CHAR2	User characterization #2
CHAR3	User characterization #3
CHAR4	User characterization #4
CHAR5	User characterization #5

Examples

`SENS:CORR:CCHeck ECALA`

sense2:correction:ccheck:acquire ecalb,char1

Query Syntax Not applicable

Overlapped? No
Default Not applicable

SENSe<cnum>:CORRection:CCHeck:DONE

(Write-only) Concludes the Confidence Check and sets the ECal module back into the idle state.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

Examples

SENS:CORR:CCH:DONE
sense2:correction:ccheck:done

Query Syntax Not applicable

Overlapped? No
Default Not applicable

SENSe<cnum>:CORRection:CCHeck:PARameter <Mname>

(Read-Write) Specifies an existing measurement to be used for the Confidence Check.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1
<Mname> Name of the measurement you are selecting for the confidence check. The measurement must already exist.

Examples

```
SENS:CORR:CCH:PAR 'TEST'
 'selects the measurement "test" on channel 1 for the confidence check
```

sense2:correction:ccheck:parameter 'test'
'selects the measurement "test" on channel 2 for the confidence check

Query Syntax

SENSe<cnum>:CORRection:CCHeck:PARameter?

Returns the name of the selected measurement on channel <cnum>.

Overlapped? No
Default Not applicable

SENSe<cnum>:CORRection:COLLect[:ACQuire] <class>[,subclass]

(Write-only) Measures the specified standards from the selected calibration kit. The calibration kit is selected using the Sense:Correction:Collect:CKIT command.

Note: Before using this command you must select two items:

> Select a calibration method using SENS:CORR:COLL:METH
> Select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

<class> **Measures the standards associated with these class labels:**

Choose from:

STAN1	S11A and S22A
STAN2	S11B and S22B
STAN3	S11C and S22C
STAN4	S21T and S12T - usually the THRU standard
STAN5	Generic Isolation; not associated with calibration kit definition
ECALA	ECAL module A
ECALB	ECAL module B
SLSET	Sets 'sliding load type', and increments the "number of slides" count. The total number of slides is critical to the correct calculation of the sliding load algorithm. See a sliding load cal example.
SLDONE	Computes the sliding load using a circle fit algorithm.

[subclass] Optional argument. For mechanical calibration kits, choose from the following to specifying the standard identified in the SENS:CORR:COLL:CKIT:ORDer list to be acquired. If this argument is not used, the default is **SST1**.

SST1	First standard in the order list
SST2	Second standard in the order list
SST3	Third standard in the order list
SST4	Fourth standard in the order list
SST5	Fifth standard in the order list
SST6	Sixth standard in the order list
SST7	Seventh standard in the order list

If ECALA or ECALB is specified for <class>, choose one of the following for specifying which characterization within the ECal module will be used for the acquire. If not specified, the default is **CHAR0**.

CHAR0	Factory characterization (data that was stored in the ECal module by Agilent)
CHAR1	User characterization #1
CHAR2	User characterization #2
CHAR3	User characterization #3
CHAR4	User characterization #4
CHAR5	User characterization #5

Examples

```
SENS:CORR:COLL STAN1

'If SENS:CORR:COLL:CKIT:ORDer2 5,3,7
 was specified, the following command measures standard 3
(the second in the order list)
 sense1:correction:collect:acquire stan3,sst2
```

SENS:CORR:COLL ECALA
sense2:correction:collect:acquire ecalb,char1

Query Syntax Not applicable

Overlapped? No
Default Not applicable

SENSe<cnum>:CORRection:COLLect:APPLy

(Write-only) Applies error terms to the measurement that is selected using Calc:Par:Select.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Note: This command is only necessary if you need to modify error terms. If you do not need to modify error terms, SENSe<cnum>:CORRection:COLLect:SAVE calculates and then automatically applies error terms after you use SENS:CORR:COLL:ACQuire to measure cal standards.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

Example

1. **CALCulate2:PARameter:SELect S21_2 'select the measurement to apply terms to**
2. **SENSe2:CORRection:COLLect:METHod SPARSOLT 'set type of cal method.**
3. **CALCulate2:DATA? SCORR1 'download the error term of interest**
4. **'Modify the error term here**
5. **CALCulate2:DATA SCORR1 'upload the error term of interest**

SENSe2:CORRection:COLLect:APPLy 'applies the error terms to the measurement

Query Syntax Not applicable

Overlapped? No
Default Not applicable

SENSe<cnum>:CORRection:COLLect:METHod <char>

(Read-Write) Sets the calibration method. (also known as 'Calibration Type' on calibration dialog box.)

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1
<char> Choose from:

Method	Description
NONE	No Cal method
GUIDED	Guided calibration
REFL1OPEN	Response Open
REFL1SHORT or **REFL1**	Response Short
REFL3	Full 1 port
TRAN1	Response Thru
TRAN2	Response Thru and Isolation
SPARSOLT	Full SOLT 2 port

Examples SENS:CORR:COLL:METH REFL1
sense2:correction:collect:method sparsolt

Query Syntax	SENSe<cnum>:CORRection:COLLect:METHod?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:SAVE

(Write-only) Calculates the error terms using the selected :METHod and applies the error terms to the selected measurement (turns error correction ON.) Does NOT save the calibration error-terms.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:CORR:COLL:SAVE sense2:correction:collect:save
Query Syntax	Not applicable
Overlapped?	No
Default	Not applicable

SENSe<cnum>:CORRection:EXTension:PORT<pnum>[:TIME] <num>

(Read-Write) Sets the extension value at the specified port. Must also set SENS:CORR:EXT ON.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pnum>	Number of the port that will receive the extension. If unspecified, value is set to 1. Choose from: **1** for Port 1 **2** for Port 2
<num>	The port extension in seconds; may include suffix. Choose a number between: **-10** and **10**
Examples	SENS:CORR:EXT:PORT 2MS sense2:correction:extension:port2 .00025
Query Syntax	SENSe<cnum>:CORRection:EXTension:PORT<pnum> [:TIME]?
Return Type	Character
Overlapped?	No
Default	0

SENSe<cnum>:CORRection:EXTension:RECeiver<Rnum>[:TIME] <num>

(Read-Write) Sets the extension value at the specified receiver. Must also set SENS:CORR:EXT ON**.**

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<Rnum>	Number of the receiver that will receive the extension. If unspecified, value is set to 1 Choose from: **1** for Receiver A **2** for Receiver B
<num>	The electrical length in seconds; may include suffix. Choose a number between: **-10** and **10**
Examples	SENS:CORR:EXT:REC 2MS sense2:correction:extension:receiver2:time .00025
Query Syntax	SENSe<cnum>:CORRection:EXTension:RECeiver<Rnum> [:TIME]?
Return Type	Character
Overlapped?	No
Default	0

SENSe<cnum>:CORRection:EXTension[:STATe] <ON | OFF>

(Read-Write) Turns port extensions ON or OFF.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns port entensions ON. **OFF** (or 0) - turns port extensions is OFF.
Examples	SENS:CORR:EXT ON sense2:correction:extension:state off
Query Syntax	SENSe<cnum>:CORRection:EXTension[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

SENSe:CORRection:IMPedance:INPut:MAGNitude <num>

(Read-Write) Sets and returns the system impedance value for the analyzer.

Parameters

<num>	System Impedance value in ohms. Choose any number between 0 and 1000 ohms.
Examples	SENS:CORR:IMP:INP:MAGN 75 sense:correction:impedance:input:magnitude 50.5
Query Syntax	SENSe:CORRection:IMPedance: INPut:MAGNitude?
Return Type	Character
Overlapped?	No
Default	50

SENSe<cnum>:CORRection:INTerpolate[:STATe] <ON | OFF>

(Read-Write) Turns correction interpolation ON or OFF.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns interpolation ON. **OFF** (or 0) - turns interpolation OFF.
Examples	SENS:CORR:INT ON sense2:correction:interpolate:state off
Query Syntax	SENSe<cnum>:CORRection:INTerpolate[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

SENSe<cnum>:CORRection:ISOLation[:STATe] <ON | OFF>

(Read-Write) Turns isolation cal ON or OFF during Full 2-port calibration. If this comand is not sent, the default state is to **disable** Isolation.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns isolation ON. **OFF** (or 0) - turns isolation OFF.
Examples	SENS:CORR:ISOL ON sense2:correction:isolation:state off
Query Syntax	SENSe<cnum>:CORRection:ISOLation[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF - (Isolation disabled)

SENSe:CORRection:PREFerence:ECAL:ORIentation[:STATe] <ON|OFF>

(Read-Write) Specifies whether or not the PNA should perform orientation of the ECal module during calibration. Orientation is a technique by which the PNA automatically determines which ports of the module are connected to which ports of the PNA. Orientation begins to fail at very low power levels or if there is much attenuation in the path between the PNA and the ECal module. If orientation is turned OFF, the SENS:CORR:PREF:ECAL:PMAP command must be used to specify the port connections before performing a cal.

Note: 3-port calibration with a 2-port ECal module does not yet fully support the mode of orientation = OFF.

Examples	**SENS:CORR:PREF:ECAL:ORI OFF** sense:correction:preference:ecal:orientation:state on
Query Syntax	SENSe:CORRection:PREFerence:ECAL:ORIentation[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No

Default	ON (1)

SENSe:CORRection:PREFerence:ECAL:PMAP <module>,<string>

(Read-Write) When ECal module orientation is turned OFF (SENS:CORR:PREF:ECAL:ORI OFF), this command specifies the port mapping (which ports of the module are connected to which ports of the PNA) prior to performing ECal calibrations.

Parameters

<module> Specifies which ECal module this port map is being applied to. Choose from:

ECALA ECal module A

ECALB ECal module B

<string> Format this parameter in the following manner:

Aw,Bx,Cy,Dz

where

- A, B, C, and D are literal ports on the ECAL module
- w,x,y, and z are substituted for PNA port numbers to which the ECAL module port is connected.

Ports of the module which are not used are omitted from the string.

For example, on a 4-port ECal module with

port A connected to PNA port 2

port B connected to PNA port 3

port C not connected

port D connected to PNA port 1

the string would be: A2,B3,D1

If either the receive port or source port (or load port for 2-port cal) of the CALC:PAR:SELected measurement is not in this string and orientation is OFF, an attempt to perform an ECal calibration will fail.

Examples SENS:CORR:PREF:ECAL:PMAP ECALA, 'A1,B2'
sense:correction:preference:ecal:pmap ecalb, 'a2,b1,c3'

Query Syntax SENSe:CORRection:PREFerence:ECAL:PMAP? <module>
Return Type Character

Overlapped? No
Default Null string ()

SENSe<cnum>:CORRection:RVELocity:COAX <num>

(Read-Write) Sets the velocity factor to be used with Electrical Delay and Port Extensions.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

<num> Velocity factor. Choose a number between:
0 and **10**
(.66 polyethylene dielectric; .7 teflon dielectric)

Note: to specify the electrical delay for reflection measurements (in both directions), double the velocity factor.

Examples SENS:CORR:RVEL:COAX .66

sense2:correction:rvelocity:coax .70

Query Syntax SENSe<cnum>:CORRection:RVELocity:COAX?
Return Type Character

Overlapped? No
Default 1

SENSe:CORRection:SFORward[:STATe] <boolean>

(Read-Write) Sets the direction a calibration will be performed when only one set of standards is used.

Use SENSe:CORRection:TSTandards[:STATe] **OFF** to specify that only one set of standards will be used.

Parameters

<boolean> **ON (1)** - FORWARD direction of a 2-port calibration will be performed

OFF (0) - REVERSE direction of a 2-port calibration will be performed

Examples

```
SENS:CORR:SFOR 1
 sense:correction:sforward:state 0
```

See an example using this command

Query Syntax Not applicable

Overlapped? No
Default ON

SENSe<cnum>:CORRection[:STATe] <ON | OFF>

(Read-Write) Specifies whether or not correction data is applied to the measurement.

Note: Before using this command you must select a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1
<ON | OFF> **ON** (or 1) - correction is applied to the measurement.
OFF (or 0) - correction is NOT applied to the measurement.

Examples SENS:CORR ON
sense2:correction:state off

Query Syntax SENSe<cnum>:CORRection[:STATe]?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default OFF

SENSe:CORRection:TSTandards[:STATe] <boolean>

(Read / Write) Specifies the acquisition of calibration data using TWO set of standards or ONE.

Parameters

<boolean> **ON (1)** - TWO sets of standards will be used for full 2-port calibration for both Forward and Reverse parameters.

OFF (2) - ONE set of standards will be used for full 2-port calibration.

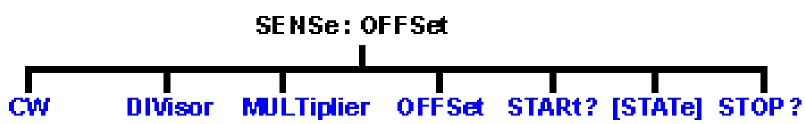
Set SENSe:CORRection:COLLect:SFORward[:STATe] to **ON** for the forward acquisitions and **OFF** for the reverse acquisitions.

Examples	`SENS:CORR:TST 1`
	`sense:correction:tstandard:state 0`
	See an example using this command

Query Syntax	SENSe:CORRection:TSTandards[:STATe]?
Overlapped?	No
Default	ON

Sense:Offset Commands

Sets the offset frequency functions, causing the stimulus and response frequencies to be different.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Frequency Offset

SENSe<cnum>:OFFSet:CW <bool>

(Read-Write) Turns stimulus CW Override mode ON or OFF. Use this setting to establish a fixed (CW) stimulus frequency while measuring the Response over a swept frequency range. Learn more about Frequency Offset.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<bool>	ON (or 1) - turns CW override ON. OFF (or 0) - turns CW overide OFF.
Examples	SENS:OFFS:CW ON sense2:offset:cw off
Query Syntax	SENSe<cnum>:OFFSet:CW?
Return Type	Boolean
Overlapped?	No
Default	OFF

SENSe<cnum>:OFFSet:DIVisor <num>

(Read-Write) Specifies (along with the multiplier) the value to multiply by the stimulus. Learn more about Frequency Offset.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1

<num>	Divisor value. Range is 1 to 1000
Examples	SENS:OFFS:DIV 3 sense2:offset:divisor 2
Query Syntax	SENSe<cnum>:OFFSet:DIVisor?
Return Type	Character
Overlapped?	No
Default	1

SENSe<cnum>:OFFSet:MULTiplier <num>

(Read-Write) Specifies (along with the divisor) the value to multiply by the stimulus. Learn more about Frequency Offset.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Multiplier value. Range is +/- 1000. Negative multipliers cause the stimulus to sweep in decreasing direction. For mixer measurements, this would be for setups requiring the RF frequency to be less than LO frequency
Examples	SENS:OFFS:MULT 2 sense2:offset:multiplier 4
Query Syntax	SENSe<cnum>:OFFSet:MULTplier?
Return Type	Character
Overlapped?	No
Default	1

SENSe<cnum>:OFFSet:OFFSet <num>

(Read-Write) Specifies an absolute offset frequency in Hz. For mixer measurements, this would be the LO frequency. Learn more about Frequency Offset.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Offset frequency. Range is +/- 1000 GHz. Offsets can be positive or negative
Examples	SENS:OFFS:OFFS 1GHz sense2:offset:offset 1e9
Query Syntax	SENSe<cnum>:OFFSet:OFFSet?
Return Type	Character
Overlapped?	No
Default	**0 Hz**

SENSe<cnum>:OFFSet:STARt?

(Read-Only) Returns the response start frequency Learn more about Frequency Offset.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:OFFS:STAR? sense2:offset:start?

Return Type Character

Overlapped? No
Default Not applicable

SENSe<cnum>:OFFSet:[STATe] <bool>

(Read-Write) Enables Frequency Offset Mode on ALL measurements that are present on the active channel. This immediately causes the source and receiver to tune to separate frequencies. The receiver frequencies are specified with the other SENS:OFFSet commands. To make the stimulus settings use the SENS:FREQ commands.

Tip: To avoid unnecessary errors, first make other offset frequency settings, then set Frequency Offset ON. Learn more about Frequency Offset.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1
<bool> ON (or 1) - turns Frequency Offset ON.
OFF (or 0) - turns Frequency Offset OFF.

Examples SENS:OFFS ON
sense2:offset:state off

Query Syntax SENSe<cnum>:OFFSet:[STATe]?
Return Type Boolean

Overlapped? No
Default OFF (0)

SENSe<cnum>:OFFSet:STOP?

(Read-Only) Returns the response stop frequency. Learn more about Frequency Offset.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

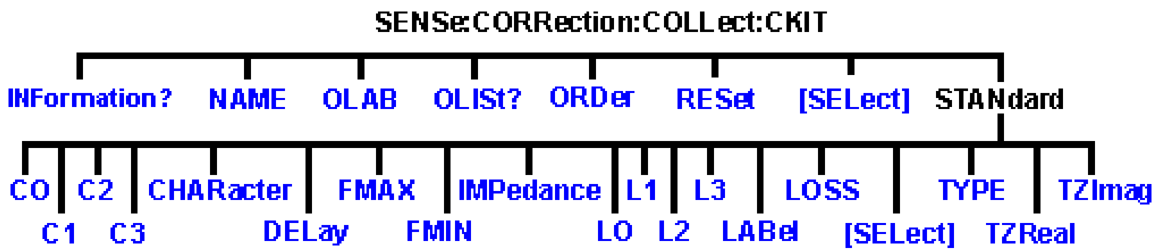
Examples SENS:OFFS:STOP
sense2:offset:stop

Return Type Character

Overlapped? No
Default Not applicable

Sense:Correction:Collect:CKit Commands

Use to change the definitions of calibration kit standards.



Most of these commands act on the currently selected standard from the currently selected calibration kit.

- To select a Calibration kit, use SENS:CORR:COLL:CKIT:SEL.
- To select a Calibration standard, use SENS:CORR:COLL:CKIT:STAN:SEL
- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Modifying Cal Kits

Note: You should provide data for every definition field - for every standard in your calibration kit. If a field is not set, the default value may not be what you expect.

SENSe:CORRection:COLLect:CKIT:INFormation? <module>[,char]

(Read Only) Reads characterization information from an ECal module. This command returns the same string as the GetECALModuleInfo method on the Calibrator COM object.

Parameters

<module> Specifies which ECal module to read from. Choose from:

ECALA ECal module A

ECALB ECal module B

[char] Optional argument.

Specifies which characterization within the ECal module to read information from. If this argument is not used, the default is **CHAR0**. **CHAR1** through **CHAR5** are for user characterizations that may have been written to the module by the User Characterization feature on the PNA. Choose from:

CHAR0 Factory characterization (data that was stored in the ECal module by Agilent)

CHAR1 User characterization #1

CHAR2 User characterization #2

CHAR3 User characterization #3

CHAR4 User characterization #4

CHAR5 User characterization #5

Examples

SENS:CORR:COLL:CKIT:INF? ECALA

sense:correction:collect:ckit:information? ecalb,char1

Example return string:

ModelNumber: 85092-60007, SerialNumber: 01386, ConnectorType: N5FN5F, PortAConnector: Type N (50) female, PortBConnector: Type N (50) female, MinFreq: 30000, MaxFreq: 9100000000, NumberOfPoints: 250, Calibrated: July 4 2002

Return Type Character

Overlapped? No

Default Not Applicable

SENSe:CORRection:COLLect:CKIT:NAME <name>

(Read-Write) Sets a name for the selected calibration kit.

Parameters

<name> Calibration Kit name. Any string name, can include numerics, period, and spaces; any length (although the dialog box display is limited to about 30 characters).

Examples SENS:CORR:COLL:CKIT:NAME 'MYAPC35'
sense:correction:collect:ckit:name 'mytypen'

Query Syntax SENSe:CORRection:COLLect:CKIT:NAME?

Return Type String

Overlapped? No

Default Not Applicable

SENSe:CORRection:COLLect:CKIT:OLABel<class> <name>

(Read-Write) Sets the label for the calibration class designed by <class>. The label is used in the prompts for connecting the calibration standards associated with that <class>.

Parameters

<class> Number of the calibration class. Choose a number between: 1 and 18.
The <class> numbers are associated with the following calibration Classes:

<class>	**Class**	**Description**
1	S11A	Reflection standard
2	S11B	Reflection standard
3	S11C	Reflection standard
4	S21T	Thru/Delay standard
5	S22A	Reflection standard
6	S22B	Reflection standard
7	S22C	Reflection standard
8	S12T	Thru/Delay standard
3-port analyzers only		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru/Delay standard
13	S23T	Thru/Delay standard
14	S31T	Thru/Delay standard
15	S13T	Thru/Delay standard
	TRL Calibrations	
16	TRL "T"	Thru standard
17	TRL "R"	Reflect standard
18	TRL "L"	Thru standard

<name>	Label for the calibration class. Must be enclosed in quotes. Any string between 1 and 12 characters long. Cannot begin with a numeric.

Examples

`SENS:CORR:COLL:CKIT:OLAB3 'LOADS'`

sense:correction:collect:ckit:olabel4 'Thru'

Return Type String

Overlapped? No
Default Not Applicable

SENSe:CORRection:COLLect:CKIT:OLIST[class]?

(Read-only) Returns seven values of standards that are assigned to the specified class.

Parameters

<class> Number of the calibration class to be queried. The <class> numbers are associated with the following calibration Classes:

<class>	**Class**	**Description**
1	S11A	Reflection standard
2	S11B	Reflection standard
3	S11C	Reflection standard
4	S21T	Thru/Delay standard
5	S22A	Reflection standard
6	S22B	Reflection standard
7	S22C	Reflection standard
8	S12T	Thru/Delay standard
3-port analyzers only		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru/Delay standard
13	S23T	Thru/Delay standard
14	S31T	Thru/Delay standard
15	S13T	Thru/Delay standard
	TRL Calibrations	
16	TRL "T"	Thru standard
17	TRL "R"	Reflect standard
18	TRL "L"	Thru standard

<class> Number of the calibration class to be queried. The <class> numbers are associated with the following calibration Classes:

Examples

SENS:CORR:COLL:CKIT:OLIST8?

Always returns 7 standard numbers. Unassigned standards return 0

Return Type Character; returns the <class> number of the selected standard.

Overlapped? No
Default Not Applicable

SENSe:CORRection:COLLect:CKIT:ORDer<class> <std> [,<std>] [,<std>] [,<std>] [,<std>] [,<std>] [,<std>]

(Read-Write) Sets a standard number to a calibration class. Does **NOT** set or dictate the order for measuring the standards. For more information, see Assigning Standards to a Calibration Class

Parameters

<class> Number of the calibration class that is assigned to <standard>. Choose a number between:

1 and **18**

The <class> numbers are associated with the following calibration Classes:

<class>	**Class**	**Description**
1	S11A	Reflection standard
2	S11B	Reflection standard
3	S11C	Reflection standard
4	S21T	Thru/Delay standard
5	S22A	Reflection standard
6	S22B	Reflection standard
7	S22C	Reflection standard
8	S12T	Thru/Delay standard
3-port analyzers only		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru/Delay standard
13	S23T	Thru/Delay standard
14	S31T	Thru/Delay standard
15	S13T	Thru/Delay standard
		TRL Calibration
16	TRL "T"	Thru standard
17	TRL "R"	Reflect standard
18	TRL "L"	Thru standard

<std> Standard number to be assigned to the class; Choose a standard between 1 and 8. One standard is mandatory; up to six additional standards are optional.

Examples

Assigns standard 3 to S11A class:

SENS:CORR:COLL:CKIT:ORD1 3

Assigns standard 2 and 5 to S21T class class:

sense:correction:collect:ckit:order4 2,5

Query Syntax

SENSe:CORRection:COLLect:CKIT:ORDer<class>?

'Returns only the first standard assigned to the specified class. To query the remaining

	standards, use SENSe:CORRection:COLLect:CKIT:OLIST[1-15]?
Return Type	Character.
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:RESet <num>

(Write-only) Resets the selected calibration kit to factory default definition values.

Parameters

<num>	The number of the calibration kit to be reset. Choose any integer between: **1** and **8**
Examples	SENS:CORR:COLL:CKIT:RESet 1 sense:correction:collect:ckit:reset 4
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT[:SELect] <num>

(Read-Write) Selects (makes active) a calibration kit for **performing** a calibration or for **modifying** standards. All subsequent "CKIT" commands that are sent apply to this selected calibration kit. Select a calibration standard using SENS:CORR:COLL:CKIT:STAN <num>

Parameters

<num> The number of the calibration kit. Choose from:

Use SENSe:CORRection:COLLect:CKIT:RESet to restore Cal Kits to default values.

<num>	Name
1	User Defined 1
2	User Defined 2
3	User Defined 3
4	User Defined 4
	”
	”
	”
48	User Defined 48
49	User Defined 49
50	User Defined 50
99	ECAL module

Examples	SENS:CORR:COLL:CKIT 2 sense2:correction:collect:ckit:select 7
Query Syntax	SENSe:CORRection:COLLect:CKIT?
Return Type	Character
Overlapped?	No

Default 1

SENSe:CORRection:COLLect:CKIT:STANdard:C0 <num>

(Read-Write) Sets the C0 value (the first capacitance value) for the selected standard.

Parameters

<num>	Value for C0 in picofarads
Examples	**The following commands set C0=15 picofarads:** SENS:CORR:COLL:CKIT:STAN:C0 15 sense:correction:collect:ckit:standard:c0 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:C0?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:C1 <num>

(Read-Write) Sets the C1 value (the second capacitance value) for the selected standard.

Parameters

<num>	Value for C1 in picofarads
Examples	The following two commands set C1=15 picofarads: SENS:CORR:COLL:CKIT:STAN:C1 15 sense:correction:collect:ckit:standard:c1 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:C1?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:C2 <num>

(Read-Write) Sets the C2 value (the third capacitance value) for the selected standard.

Parameters

<num>	Value for C2 in picofarads
Examples	The following two commands set C2=(-15) picofarads: SENS:CORR:COLL:CKIT:STAN:C2 -15 sense:correction:collect:ckit:standard:c2 -15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:C2?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:C3 <num>

(Read-Write) Sets the C3 value (the fourth capacitance value) for the selected standard.

Parameters

<num>	Value for C3 in picofarads
Examples	The following two commands set C3=15 picofarads: SENS:CORR:COLL:CKIT:STAN:C3 15 sense:correction:collect:ckit:standard:c3 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:C3?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:CHARacter <char>

Note: Character is sometimes referred to as **Medium**

(Read-Write) Sets the media type of the selected calibration standard.

Parameters

<char>	Media type of the standard. Choose from: **Coax** - Coaxial Cable **Wave** - Waveguide
Examples	SENS:CORR:COLL:CKIT:STAN:CHAR COAX sense:correction:collect:ckit:standard:character wave
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:CHARacter?
Return Type	Character
Overlapped?	No
Default	Coax

SENSe:CORRection:COLLect:CKIT:STANdard:DELay <num>

(Read-Write) Sets the electrical delay value for the selected standard.

Parameters

<num>	Electrical delay in seconds
Examples	SENS:CORR:COLL:CKIT:STAN:DEL 50e-12 sense2:correction:collect:ckit:standard:delay 50ps
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:DELay?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:FMAX <num>

(Read-Write) Sets the maximum frequency for the selected standard.

Parameters

<num>	Maximum frequency in Hertz.

Examples	SENS:CORR:COLL:CKIT:STAN:FMAX 9e9 sense:correction:collect:ckit:standard:fmax 9Ghz
Query Syntax **Return Type**	SENSe:CORRection:COLLect:CKIT:STANdard:FMAX? Character
Overlapped? **Default**	No Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:FMIN <num>

(Read-Write) Sets the minumum frequency for the selected standard.

Parameters

<num>	Minimum frequency in Hertz.
Examples	SENS:CORR:COLL:CKIT:STAN:FMIN 1e3 sense:correction:collect:ckit:standard:fmin 1khz
Query Syntax **Return Type**	SENSe:CORRection:COLLect:CKIT:STANdard:FMIN? Character
Overlapped? **Default**	No Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:IMPedance <num>

Note: Impedance is sometimes referred to as **Z0**

(Read-Write) Sets the characteristic impedance for the selected standard.

Parameters

<num>	Impedance in Ohms
Examples	SENS:CORR:COLL:CKIT:STAN:IMP 75 sense:correction:collect:ckit:standard:impedance 50.3
Query Syntax **Return Type**	SENSe:CORRection:COLLect:CKIT:STANdard:IMPedance? Character
Overlapped? **Default**	No 50

SENSe:CORRection:COLLect:CKIT:STANdard:L0 <num>

(Read-Write) Sets the L0 value (the first inductance value) for the selected standard.

Parameters

<num>	Value for L0 in picohenries
Examples	The following two commands set L0=15 picohenries: SENS:CORR:COLL:CKIT:STAN:L0 15 sense:correction:collect:ckit:standard:l0 15
Query Syntax **Return Type**	SENSe:CORRection:COLLect:CKIT:STANdard:L0? Character
Overlapped?	No

Default Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:L1 <num>

(Read-Write) Sets the L1 value (the second inductance value) for the selected standard.

Parameters

<num>	Value for L1 in picohenries
Examples	The following two commands set L1=15 picohenries: SENS:CORR:COLL:CKIT:STAN:L1 15 sense:correction:collect:ckit:standard:l1 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:L1?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:L2 <num>

(Read-Write) Sets the L2 value (the third inductance value) for the selected standard.

Parameters

<num>	Value for L2 in picohenries
Examples	The following two commands set L2=15 picohenries: SENS:CORR:COLL:CKIT:STAN:L2 15 sense:correction:collect:ckit:standard:l2 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:L2?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:L3 <num>

(Read-Write) Sets the L3 value (the fourth inductance value) for the selected standard.

Parameters

<num>	Value for L3 in picohenries
Examples	The following two commands set L3=15 picohenries: SENS:CORR:COLL:CKIT:STAN:L3 15 sense:correction:collect:ckit:standard:l3 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:L3?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:LABel <name>

(Read-Write) Sets the label for the selected standard. The label is used to prompt the user to connect the specified standard.

Parameters

<name>	Label for the standard; Must be enclosed in quotes. Any string between **1** and **12** characters long. Cannot begin with a numeric.
Examples	SENS:CORR:COLL:CKIT:STAN:LAB 'OPEN' sense:correction:collect:ckit:standard:label 'Short2'
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:LABel?
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:LOSS <num>

(Read-Write) Sets the insertion loss for the selected standard.

Parameters

<num>	Insertion loss in Mohms / sec. (MegaOhms per second of electrical delay)
Examples	SENS:CORR:COLL:CKIT:STAN:LOSS 3.5e9 sense:correction:collect:ckit:standard:loss 3
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:LOSS?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard[:SELECT] <num>

(Read-Write) Selects the calibration standard. All subsequent "CKIT" commands to modify a standard will apply to the selected standard. Select a calibration kit using SENS:CORR:COLL:CKIT:SEL

Parameters

<num>	Number of the standard. Choose any number between: **1** and **30**
Examples	SENS:CORR:COLL:CKIT:STAN 3 sense:correction:collect:ckit:standard:select 8
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard[:SELect]?
Return Type	Character
Overlapped?	No
Default	1

SENSe:CORRection:COLLect:CKIT:STANdard:TYPE <char>

(Read-Write) Sets the type for the selected standard.

Parameters

<char>	Choose from: **OPEN** **SHORT**

LOAD
SLOAD (sliding load)

THRU (through)

ARBI(arbitrary)

Examples	SENS:CORR:COLL:CKIT:STAN:TYPE LOAD sense:correction:collect:ckit:standard:type short
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:TYPE?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:TZReal <num>

(Read-Write) Sets the TZReal component value of the Terminal Impedance for the selected standard.

Note: Only applicable when the Standard Type is set to **ARBI**

Parameters

<num>	Value for TZReal in Ohms
Examples	**The following commands set TZReal=15 Ohms:** SENS:CORR:COLL:CKIT:STAN:TZReal 15 sense:correction:collect:ckit:standard:TZReal 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:TZReal?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect:CKIT:STANdard:TZImag <num>

(Read-Write) Sets the TZImag component value of the Terminal Impedance for the selected standard.

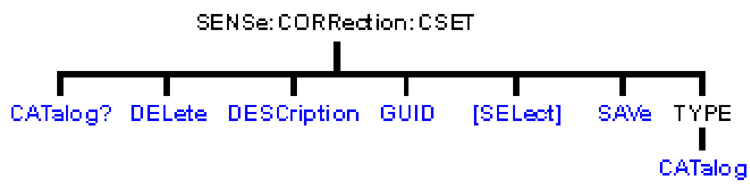
Note: Only applicable when the Standard Type is set to **ARBI**

Parameters

<num>	Value for TZImag in Ohms
Examples	The following two commands set TZImag=15 Ohms: SENS:CORR:COLL:CKIT:STAN:TZImag 15 sense:correction:collect:ckit:standard:TZImag 15
Query Syntax	SENSe:CORRection:COLLect:CKIT:STANdard:TZImag?
Return Type	Character
Overlapped?	No
Default	Not Applicable

Sense:Correction:CSET Commands

Performs actions on calibration sets.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Measurement Calibration

SENSe<cnum>:CORRection:CSET:CATalog?

(Read-only) Returns a string containing a list of comma-separated GUIDs for Cal Sets in the following format:

{FD6F863E-9719-11d5-8D6C-00108334AE96},
{1B03B2CE-971A-11d5-8D6C-00108334AE96},
{2B893E7A-971A-11d5-8D6C-00108334AE96}

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:CORR:CSET:CAT? sense2:correction:cset:catalog?
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:CSET:DELete <string>

(Write-only) Deletes a Cal Set from the set of available Cal Sets. This command immediately updates the Cal Set file on the hard drive. Using the Cal Sets collection is a convenient way to manage Cal Sets.

If the Cal Set identified by the GUID is currently in use, the Cal Set will not be deleted. If you still want to delete a Cal Set that is in use, either turn off correction on the subscribing measurement, turn off subscribed channels, or select a different Cal Set for the subscribed channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<string>	The GUID of the Cal Set to be deleted. The curly brackets and hyphens must be included. Not case sensitive.
Examples	SENS:CORR:CSET:DEL '{2B893E7A-971A-11d5-8D6C-00108334AE96}' sense2:correction:cset:delete '{2B893E7A-971A-11d5-8D6C-00108334AE96}'
Query Syntax	Not Applicable

Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:CSET:DESCription <string>

(Read-Write) Sets or returns the descriptive string assigned to the selected Cal Set. Change this string so that you can easily identify each Cal Set. Select the Cal Set using SENSe:CORRection:CSET:GUID

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<string>	The descriptive string associated with the currently-selected Cal Set
Examples	SENS:CORR:CSET:DESC 'MyCalSet' sense2:correction:cset:description 'thisCalSet'
Query Syntax	SENSe<cnum>:CORRection:CSET:DESCription?
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:CSET:GUID <string>

(Read-Write) Selects the Cal Set identified by the string parameter (GUID) and applies it to the specified channel.

A Cal Set cannot be selected for a channel which is not On.

If the stimulus settings of the selected Cal Set differ from those of the selected channel, the instrument will automatically change the channel's settings to match the Cal Set.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<string>	GUID of the desired Cal Set. The curly brackets and hyphens must be included.
Examples	SENS:CORR:CSET:GUID '{2B893E7A-971A-11d5-8D6C-00108334AE96}' sense2:correction:cset:guid '{2B893E7A-971A-11d5-8D6C-00108334AE96}'
Query Syntax	SENSe<cnum>:CORRection:CSET:GUID? Returns the GUID of the currently-selected Cal Set for the specified channel.
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:CSET[:SELect] <char>

(Read-Write) Restores a correction data set from memory. The file name is

"**CSET*x*.cst"** where x is the user number assigned to <char>, and .cst specifies a cal set and instrument state. This is not the same syntax as a file saved through the default choices from the front panel, which is "**at00*x*.cst**". For more information on the file naming syntax, see the MMEMory subsystem.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: DEF - Presets the analyzer USER01- Restores User01 calibration data USER02 - Restores User02 calibration data through... USER10 - Restores User10 calibration data
Examples	SENS:CORR:CSET DEF sense2:correction:cset:select user02
Query Syntax	SENSe<cnum>:CORRection:CSET[:SELect]?
Return Type	Character
Overlapped?	No
Default	DEF

SENSe<cnum>:CORRection:CSET:SAVE <char>

Write a correction data set to memory or Read the last correction set saved. The file name is saved as "**CSET*x*.cst"** where x is the user number assigned to <char>, and .cst specifies a cal set and instrument state. This is not the same syntax as a file saved through the default choices from the front panel, which is "**at00*x*.cst**". For more information on the filenaming syntax, see the MMEMory subsystem.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: USER01 USER02... USER10
Examples	SENS:CORR:CSET:SAVE USER03 sense2:correction:cset:save user09
Query Syntax	SENSe<cnum>:CORRection:CSET:SAVE? Queries the last correction set saved.
Return Type	Character
Overlapped?	No
Default	Not applicable

SENSe<ch>:CORRection:CSET:TYPE:CATalog?<optional enum>

(Read-only) Query the caltypes available in the selected calset. The user can specify the output format: a comma separated list of guids or a list of names. See extra note

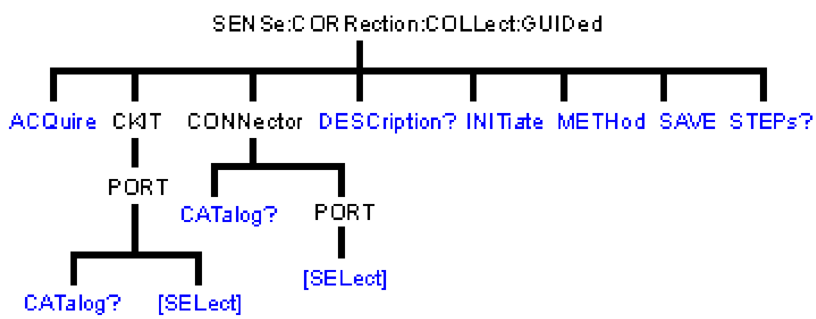
Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<optional enum>	**NAME:** (default) returns the string name of the caltype **GUID:** Returns the guid of the caltype
Examples	SENS:CORR:CSET:TYPE:CAT SENS2:CORRection:CSET:TYPE:CAT?
Query Syntax	Not Applicable
Return Type	**string**
Overlapped?	No
Default	Not Applicable

Note: CalTypes in earlier versions of PNA (pre 3.0) were indicated by a pre defined enumeration. In 3.0 the identity of the CalType has been expanded to support runtime detectable contribution calibration types. CalTypes are now identified by GUID's (128 bit number) and a name (string).

Sense:Correction:Collect:Guided Commands

Performs and applies a GUIDED measurement calibration and other error correction features.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands.
- Learn about Measurement Calibration

SENSe<cnum>:CORRection:COLLect:GUIDed:ACQuire <std>

(Write-only) Initiates the measurement of the specified calibration standard
Executing this command with an unnecessary standard has no affect.

The measured data is stored and used for subsequent calculations of error correction coefficients. All standards must be measured before a calibration can be completed. Any measurement can be repeated until the SENS:CORR:COLL:GUID:SAVE command is executed.

Query the user prompt description using SENS:CORR:COLL:GUID:DESC?
Query the required calibration steps using SENS:CORR:COLL:GUID:STEP?

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<std>	Choose from:STAN1, STAN2, STAN3, through STAN40
Examples	SENS:CORR:COLL:GUID:ACQ STAN1 sense2:correction:collect:guided:acquire stan1
Query Syntax	Not Applicable
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:CKIT:PORT<pnum>:CATalog?

(Read-only) Returns a comma-separated list of valid kits for each port. In addition to mechanical calibration kits, this will include applicable characterizations found within ECal modules currently connected to the PNA. Use items in the list to select the kit to be used with the SENS:CORR:COLL:GUID:CKIT:PORT command.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pnum>	Any existing port number: 1,2 or 3 (for 3-port analyzers). If unspecified, value is set to 1
Examples	SENS:CORR:COLL:GUID:CKIT:PORT1:CAT? 'When "Type N (50) male" is specified for connector type, returns: "85054D, 85032F"
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:CKIT:PORT<pnum>[:SELect] <kit>

(Read-Write) Specifies the calibration kit for each port to be used during a guided calibration. An unused port does NOT need to have a specified Cal Kit.

Note:

1. Specify the connector type for the port with SENS:CORR:COLL:GUID:CONN:PORT.
2. Query the valid available kits for each port with SENS:CORR:COLL:GUID:CKIT:PORT:CAT?
3. Specify the kit using this command.
4. Perform a query of this command. If the <kit> parameter was incorrectly entered, an error will be returned.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pnum>	Any existing port number: 1,2 or 3 (for 3-port analyzers). If unspecified, value is set to 1
<kit>	Calibration kit to be used for the specified port.
Examples	**SENS:CORR:COLL:GUID:CKIT:PORT1 '85055A'** sense2:correction:collect:ckit:port2:select '85092-60010 User 1 ECal'

Query Syntax	SENSe:CORRection:COLLect:GUIDed:CKIT:PORT<pnum>[:SELect]?
Return Type	String - If the <kit> parameter was incorrectly entered while writing, an error will be returned.
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:GUIDed:CONNector:CATalog?

(Read only) Returns a list of valid connectors based on the connector descriptions of the available cal kits. Use an item from the returned list to specify a connector for SENS:CORR:COLL:GUID:CONN:PORT

Parameters

none

Examples	**SENS:CORR:COLL:GUID:CONN:CAT?** **Returns:** Type N (50) female, Type N (50) male, APC 7 (50), 3.5 mm (50) male, 3.5 mm (50) female, User Connector A
Query Syntax	Not Applicable
Return Type	**string:** comma separated string values
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:CONNector:PORT<pnum>[:SELect] <conn>

(Read-Write) Specifies a connector type for every port during the Guided Calibration procedure. Valid connector names are stored within calibration kits. Some cal kits may include both male and female connectors. Therefore, specifying connector gender may be required.

Unused ports must be defined as or Not used. If all ports are defined as "Not used", a guided calibration cannot be performed.

- A single port with a valid <conn> name indicates a 1-Port calibration will be performed.
- Two ports with valid <conn> names indicate either a 2-Port or TRL calibration will be performed depending on the standards definition found within the cal kit and the capability of the analyzer. (The analyzer must have 4 receivers for TRL calibrations.).
- Three ports with valid <conn> names indicate a 3-Port calibration will be performed.

Note:

1. Use SENS:CORR:COLL:GUID:CONN:CAT? to query available connectors before specifying the port connector.
2. Select a connector type using this command.
3. Perform a query of this command. If the <conn> parameter was incorrectly entered, an error will be returned.
4. Specify the cal kit to use for each port with SENS:CORR:COLL:GUID:CKIT:PORT

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pnum>	Any existing port number: 1,2 or 3 (for 3-port analyzers). If unspecified, value is set to 1
<conn>	DUT connector type to connect with analyzer port <pnum> Some kits may include both male and female connectors so specifying

	gender may be required.
	Valid connector names are stored within calibration kits. Query available connectors using SENSe:CORRection:COLLect:GUIDed:CONNector:CATalog?
Examples	`SENS:CORR:COLL:GUID:CONN:PORT1 'Type N (50) female'` 'Indicates the DUT port that connects with the analyzer's Port1 is a TypeN 50 ohm Female connector.
Query Syntax	SENSe<cnum>:CORRection:COLLect:GUIDed:CONNector:PORT<pnum>[:SELect]?
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:DESCription? <step>

(Read-only) Returns the connection description for the specified calibration step.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<step>	A number from 1 to the number of steps required to complete the calibration (Use SENS:CORR:COLL:GUID:STEP? to query the number of steps)
Examples	`SENS:CORR:COLL:GUID:DESC ? 10` 'Returns: Connect APC 7 Open to port3
Return Type	String
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:INITiate [GUID [,bool]]

(Write-only) Initiates a guided calibration. Either create a new cal set or optionally add to / overwrite a specified cal set.

The PNA determines the measurements needed to perform the calibration using the settings specified from the SENS:CORR:COLL:GUID:CONN:PORT and SENS:CORR:COLL:GUID:CKIT:PORT commands.

After this command is executed, subsequent commands can be used to query the number of measurement steps, issue the acquisition commands, query the connection description strings, and subsequently complete a guided calibration.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<GUID>	Optional argument. If not specified a new calset is created. Calset GUID in the form: "{GUID}"; including quotes and curly brackets. The guided cal that is being initiated either supplements the existing cal set, or overwrites the cal set depending on the method, connectors, and ports selected. Learn more about Cal Sets. Must be a valid GUID; an error is reported if the GUID is not found. Query all Cal Set GUIDs with SENS:CORR:CSET:CAT?

<bool>	Optional argument. **False (0)** If cal set stimulus settings differ from the existing channel, do not change channel stimulus settings. Return an error. **This is the default setting if not specified.** **True (1)** If cal set stimulus settings differ from the existing channel, change the channel stimulus settings to match the cal set settings..
Examples	SENS:CORR:COLL:GUID:INIT "{2B893E7A-971A-11d5-8D6C-00108334AE96}",1 sense2:correction:collect:guided:initiate
Query Syntax	Not Applicable
Overlapped? **Default**	No Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:METHod <char>

(Read-Write) Selects from one of several algorithms available for performing a guided calibration.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	**Note:** to avoid errors, type the following <char> in the format shown in boldface, example use UNKN and not UNKNown. **DEFAULT** - **I**nforms guided calibrations to use the default algorithm when computing the number of needed standards acquisition steps. (In this release, the default algorithm is ADAPTER REMOVAL). **ADAP**remove - Use the adapter removal algorithm **FLUSH** - When ECal calkits are specified, use the FLUSH THRU algorithm. This selection has no affect if ECal calkits are not used or if the ECal module selected is not insertable. **UNKN**own - Use the Unknown THRU algorithm for 2-Port calibrations for non-insertable devices. This selection is not available on instruments which do not have 4 receivers. **TRL** - Select TRL caltype for 2-Port guided cals. Valid for "TRL ready" calkits with properly assigned TRL cal classes. **SOLT** - Select SOLT caltype for 2-Port guided cals. Valid for any kit with properly assigned SOLT cal classes.
Examples	SENS:CORR:COLL:GUID:METH sense2:correction:collect:guided:method unkn
Query Syntax	Not Applicable
Overlapped? **Default**	No Not Applicable

SENSe<cnum>:CORRection:COLLect:GUIDed:SAVE

(Write-only) Completes the guided cal by computing the error correction terms, turning Correction ON, and saving the calibration to a cal set.

If all of the required standards have not been measured, the calibration will not complete properly.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:CORR:COLL:GUID:SAVE sense2:correction:collect:guided:save
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

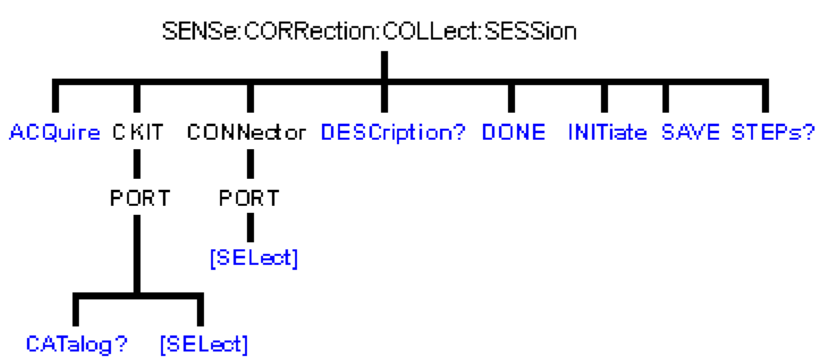
SENSe<cnum>:CORRection:COLLect:GUIDed:STEPs?

(Read-only) Returns the number of measurement steps required to complete the current guided calibration. This command is sent after the SENS:CORR:COLL:GUID:INIT, SENS:CORR:COLL:GUID:CONN:PORT and SENS:CORR:COLL:GUID:CKIT:PORT commands.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:CORR:COLL:GUID:STEP? sense2:correction:collect:guided:steps?
Return Type	Integer
Overlapped?	No
Default	Not Applicable

SENSe:CORRection:COLLect: SESSion Commands



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.

- Learn about a Calibration Session

SENSe<ch>:CORRection:COLLect:SESSion<n>:INITiate <string>

(Write) This command initiates a calibration session by locating and constructing the calibration object and querying the object to see if it supports the ICalibrate interface. If so, a valid interface handle is acquired and is associated with the session number. Note that the session number is embedded in each of the commands in the SESSion block. The default session number is "1".

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)
<string>	**string value** specifying one of the following: a) GUID: The CLSID for the custom cal type. b) Prog ID: ProgID for the custom cal type. c) Name of the custom calibration: The registered string name of custom calibration dll

Examples	SENS1:CORR:COLL:SESS6:INITiate "VectorMixerCal.VCMCType"
Query Syntax	Not Applicable
Return Type	None
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:STEPs?

(Read-only) Query the number of steps required by the caltype.

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)

Examples	SENS1:CORR:COLL:SESS6:STEPs?
Query Syntax	SENSe<ch>:CORRection:COLLect:SESSion<n>:STEPs?
Return Type	**std_int16**
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:DESC? <step>

(Read-only) Returns the connection prompt for the step. The range of <step> is limited by the results of the query for SENS:CORR:COLL:SESS:STEPS?.

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)

<step> **int** specifying the step number

Examples SENS1:CORR:COLL:SESS6:DESC?3

Query Syntax SENSe<ch>:CORRection:COLLect:SESSion<n>:DESC? <step>
Return Type **std_string**
Overlapped? No
Default Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:ACQ <step>

(Write) Acquire step. The range of <step> is limited by the results of the query for SENS:CORR:COLL:SESS:STEPS?.

Parameters

<ch> Any existing channel number. If unspecified, value is set to 1
<n> **int** specifying session number (valid numbers 1 - 16)
<step> **int** specifying the step number

Examples SENSe2:CORR:COLL:SESS6:ACQ 5

Query Syntax Not Applicable
Return Type **std_string**
Overlapped? No
Default Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:SAVE?

(Read-Write) Finish the cal - compute terms, populate and save the calset.

Parameters

<ch> Any existing channel number. If unspecified, value is set to 1
<n> **int** specifying session number (valid numbers 1 - 16)

Examples SENS1:CORR:COLL:SESS6:SAVE?

Query Syntax SENSe<ch>:CORRection:COLLect:SESSion<n>:SAVE?
Return Type **string** specifying the GUID of the calset produced by this session.
Overlapped? No
Default Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:DONE

(Write) Terminate the session, release the custom cal object. Does not save the results (see SAVE?)

Parameters

<ch> Any existing channel number. If unspecified, value is set to 1
<n> **int** specifying session number (valid numbers 1 - 16)

Examples	SENS1:CORR:COLL:SESS6:DONE
Query Syntax	Not Applicable
Return Type	None
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:CONN:PORT<p>[:SEL] <conn>

(Read-Write) Set the connector type and sex for the port number **<p>**. Connector types are discoverable using the SENS:CORR:COLL:GUID:CONN:CAT?

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)
<p>	**port number**
<conn>	**string** name of the connector type
Examples	SENS:CORR:COLL:SESS:CONN:PORT:SEL SENS2:CORR:COLL:SESS6:PORT1:SEL N Type
Query Syntax	SENS<ch>:CORR:COLL:SESS<n>:CONN:PORT<p>[:SEL] ? Query form: returns the selected connector type by string
Return Type	**string**
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:CKIT:PORT<p>:CATalog?

(Read) Query for a list of cal kits that are compatible with the connector on port <p>. The port connector type is set with SENS:CORR:COLL:SESS:PORT:SEL.

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)
<p>	**port number**
Examples	SENS2:CORR:COLL:SESS6:CKIT:PORT2:CAT?
Query Syntax	SENSe<ch>:CORRection:COLLect:SESSion<n>:CKIT:PORT<p> :CATalog?
Return Type	comma separated string values
Overlapped?	No
Default	Not Applicable

SENSe<ch>:CORRection:COLLect:SESSion<n>:CKIT:PORT<p>:SEL <calkit>

(Read-Write) Select or query the calkit for the port specified by p.

Parameters

<ch>	Any existing channel number. If unspecified, value is set to 1
<n>	**int** specifying session number (valid numbers 1 - 16)
<p>	**port number**
<calkit>	**string** name of the calkit type
Examples	SENS:CORR:COLL:SESS:CKIT:PORT:SEL 85091A SENS2:CORR:COLL:SESS6:CKIT:PORT2:SEL?
Query Syntax	SENS:CORR:COLL:SESS<n>:CKIT:PORT<p>[:SEL] ?
Return Type	comma separated string values
Overlapped?	No
Default	Not Applicable

Sense:Couple Command

Learn about Alternate Sweep

SENSe<cnum>:COUPle <ALL | NONE>

(Read-Write) Sets the sweep mode as Chopped or Alternate.

Parameters

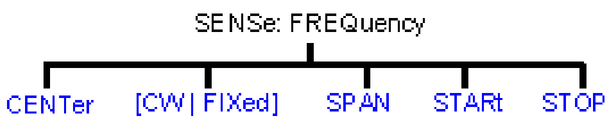
<cnum>	Any existing channel number; if unspecified, value is set to 1.
<ALL | NONE>	**ALL** - Sweep mode set to Chopped - reflection and transmission measured on the same sweep. **NONE** - Sweep mode set to Alternate - reflection and transmission measured on separate sweeps. Improves Mixer bounce and Isolation measurements. Increases sweep time
Examples	SENS:COUP ALL

sense2:couple none

Query Syntax	SENSe<cnum>:COUPle?
Return Type	Character
Overlapped?	No
Default	ALL

Sense:Frequency Commands

Sets the frequency sweep functions of the analyzer.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example using some of these commands.
- Learn about Frequency Sweep

SENSe<cnum>:FREQuency:CENTer <num>

(Read-Write) Sets the center frequency of the analyzer.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Center frequency. Choose any number between the **minimum** and **maximum** frequency limits of the analyzer. Units are Hz **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:FREQ:CENT 1000000 sense2:frequency:center 1mhz
Query Syntax	SENSe<cnum>:FREQuency:CENTer?
Return Type	Character
Overlapped?	No
Default	Center of the analyzer's frequency span

SENSe<cnum>:FREQuency[:CW |:FIXed] <num>

(Read-Write) Sets the Continuous Wave (or Fixed) frequency. Must also send SENS:SWEEP:TYPE CW to put the analyzer into CW sweep mode.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1

<num>	CW frequency. Choose any number between the **minimum** and **maximum** frequency limits of the analyzer. Units are Hz. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:FREQ 1000000 SENS:FREQ:CW MIN sense2:frequency:fixed 1mhz
Query Syntax	SENSe<cnum>:FREQuency[:CW I :FIXed]?
Return Type	Character
Overlapped?	No
Default	1 GHz

SENSe<cnum>:FREQuency:SPAN <num>

(Read-Write) Sets the frequency span of the analyzer.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Frequency span. Choose any number between: **0** (minimum) and the **maximum** frequency span of the analyzer. Units are Hz **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:FREQ:SPAN 1000000 sense2:frequency:span max
Query Syntax	SENSe<cnum>:FREQuency:SPAN?
Return Type	Character
Overlapped?	No
Default	Maximum frequency span of the analyzer

SENSe<cnum>:FREQuency:STARt <num>

(Read-Write) Sets the start frequency of the analyzer.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Start frequency. Choose any number between the **MIN** and **MAX** frequency limits of the analyzer. Units are Hz **Note**: If FREQ:START is set greater than FREQ:STOP, then STOP is set equal to START. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:FREQ:STAR 1000000 sense2:frequency:start MIN
Query Syntax	SENSe<cnum>:FREQuency:STARt?
Return Type	Character
Overlapped?	No
Default	Minimum frequency of the analyzer

SENSe<cnum>:FREQuency:STOP <num>

(Read-Write) Sets the stop frequency of the analyzer.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Stop frequency. Choose any number between: the **minimum** and **maximum** frequency limits of the analyzer. Units are Hz
	If FREQ:STOP is set less than FREQ:START, then START will be set equal to STOP.
	Note: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:FREQ:STOP 1000000 sense2:frequency:stop max
Query Syntax	SENSe<cnum>:FREQuency:STOP?
Return Type	Character
Overlapped?	No
Default	Maximum frequency of the analyzer

Sense:Power Command

Learn about Receiver Attenuation

SENSe<cnum>:POWer:ATTenuation <recvr>,<num>

(Read-Write) Sets the attenuation level for the specified receiver.

Note: Attenuation cannot be set with Sweep Type set to Power

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<recvr>	Receiver to get attenuation. Choose from: **ARECeiver** - receiver A **BRECeiver** - receiver B
<num>	Choose from: **0** to **35** dB - in 5 dB steps If a number other than these is entered, the analyzer will select the next lower valid value. For example, if 19.9 is entered for <num> the analyzer will switch in 15 dB attenuation.
Examples	SENS:POW:ATT AREC,10 sense2:power: attentuation breceiver,30
Query Syntax	SENSe<cnum>:POWer :ATTenuation? <rec>
Return Type	Character

Overlapped?	No
Default	0

Sense:Roscillator Command

Learn about the Reference Osc.

SENSe:ROSCillator:SOURce?

(Read-only) Applying a signal to the Reference Oscillator connector automatically sets the Reference Oscillator to EXTernal. This command allows you to check that it worked.
EXT is returned when a signal is present at the **Reference Oscillator** connector.
INT is returned when **NO** signal is present at the **Reference Oscillator** connector.

Examples	SENS:ROSC:SOUR? sense:roscillator:source?
Return Type	Character
Overlapped?	No
Default	Not applicable

Route Command

Learn about Frequency Offset

ROUTe:PATH:LOOP:R1 <char>

(Read-Write) Throws internal switch to reference receiver. This feature is only available on PNA models with Option 081 - external reference switch. See block diagram of the reference switch.

Parameters

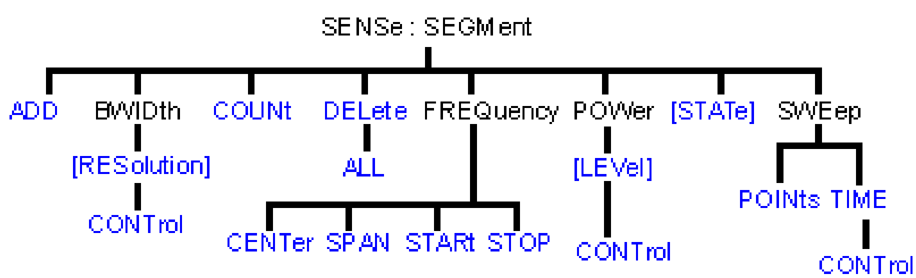
<char>	Position of the switch. Choose from: **INTernal** - bypass R1 Loop. Connects the port 1 source directly to the R1 receiver. **EXTernal** - flow through R1 Loop. Allows direct access to the R1 receiver through the Reference 1 front-panel connectors.
Examples	ROUT:PATH:LOOP:R1 INT

route:path:loop:r1 external

Query Syntax	ROUTe:PATH:LOOP:R1?
Return Type	Character
Overlapped?	No
Default	INTernal

Sense:Segment Commands

Defines the segment sweep settings. Enable segment sweep with **SENS:SWE:TYPE SEGMent.**



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Segment Sweep

SENSe<cnum>:SEGMent<snum>:ADD

(Write-only) Adds a segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to add. If unspecified, value is set to 1. Segment numbers must be sequential. If a new number is added where one currently exists, the existing segment and those following are incremented by one.
Examples	Two Segments exist (1 and 2). The following command will add a new segment (1). The existing (1 and 2) will become (2 and 3) respectively**.** SENS:SEGM1:ADD sense2:segment1:add
Query Syntax	Not applicable. Use Sense:Segment:Count to determine the number of segments in a trace.
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:SEGMent<snum>:BWIDth[:RESolution] <num>

(Read-Write) Sets the IFBandwidth for the specified segment. First set SENS:SEGM:BWIDth:CONTrol ON. All subsequent segments that are added assume the new IF Bandwidth value.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<num>	IF Bandwidth. Choose from: **1 | 2 | 3 | 5 | 7 | 10 | 15 | 20 | 30 | 50 | 70 | 100 | 150 | 200 | 300 | 500 | 700 | 1k | 1.5k | 2k | 3k | 5k | 7k | 10k | 15k | 20k | 30k | 35k | 40k |** If a number other than these is entered, the analyzer will round up to the closest valid number (unless a number higher than the maximum in entered.) **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples SENS:SEGM:BWID 1KHZ
sense2:segment2:bwidth:resolution max

Query Syntax SENSe<cnum>:SEGMent<snum>:BWIDth[:RESolution]?
Return Type Character

Overlapped? No
Default 35k

SENSe<cnum>:SEGMent:BWIDth[:RESolution]:CONTrol <ON | OFF>

(Read-Write) Specifies whether the IF Bandwidth resolution can be set independently for each segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns Bandwidth control ON. Bandwidth can be set for each segment **OFF** (or 0) - turns Bandwidth control OFF. Use channel bandwidth setting

Examples SENS:SEGM:BWID:CONT ON
sense2:segment:bwidth:control off

Query Syntax SENSe<cnum>:SEGMent:BWIDth[:RESolution]:CONTrol?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default OFF

SENSe<cnum>:SEGMent:COUNt?

(Read-only) Queries the number of segments that exist in the specified channel.

Parameters

<cnum> Any existing channel number. If unspecified, value is set to 1

Examples SENS:SEGM:COUNt?
sense2:segment:count?

Return Type Character

Overlapped? No
Default 1 segment

SENSe<cnum>:SEGMent<snum>:DELete

(Write-only) Deletes the specified sweep segment.

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Number of the segment to delete. If unspecified, value is set to 1
Examples	SENS:SEGM:DEL sense2:segment2:delete
Query Syntax	Not applicable
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:SEGMent:DELete:ALL

(Write-only) Deletes all sweep segments.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SENS:SEGM:DEL:ALL sense2:segment:delete:all
Query Syntax	Not applicable
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:SEGMent<snum>:FREQuency:CENTer <num>

(Read-Write) Sets the Center Frequency for the specified segment. The Frequency Span of the segment remains the same. The Start and Stop Frequencies change accordingly.

Note: All previous segment's Start and Stop Frequencies that are larger than the new Start Frequency are changed to the new Start Frequency. All following segment's start and stop frequencies that are smaller than the new Stop Frequency are changed to the new Stop Frequency.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<num>	Center Frequency in Hz. Choose any number between the **minimum** and **maximum** frequency of the analyzer. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SEGM:FREQ:CENT 1MHZ sense2:segment2:frequency:center 1e9
Query Syntax	SENSe<cnum>:SEGMent<snum>:FREQuency:CENTer?
Return Type	Character
Overlapped?	No
Default	Stop Frequency of the previous segment. If first segment, start frequency of the analyzer.

SENSe<cnum>:SEGMent<snum>:FREQuency:SPAN <num>

(Read-Write) Sets the Frequency Span for the specified segment. The center frequency of the segment remains the same. The start and stop frequencies change accordingly.

Note: All previous segment's Start and Stop Frequencies that are larger than the new Start

Frequency are changed to the new Start Frequency. All following segment's start and stop frequencies that are smaller than the new Stop Frequency are changed to the new Stop Frequency.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<num>	Frequency Span in Hz. Choose any number between the **minimum** and **maximum** frequency of the analyzer. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SEGM:FREQ:SPAN 1MHZ sense2:segment2:frequency:span max
Query Syntax	SENSe<cnum>:SEGMent<snum>:FREQuency:SPAN?
Return Type	Character
Overlapped?	No
Default	If first segment, frequency span of the analyzer. Otherwise 0.

SENSe<cnum>:SEGMent<snum>:FREQuency:START <num>

(Read-Write) Sets the Start Frequency for the specified sweep segment.

Note: All other segment Start and Stop Frequency values that are larger than this frequency are changed to this frequency.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<num>	Start Frequency in Hz. Choose any number between the **minimum** and **maximum** frequency of the analyzer. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SEGM:FREQ:STAR 1MHZ sense2:segment2:frequency:start minimum
Query Syntax	SENSe<cnum>:SEGMent<snum>:FREQuency:STARt?
Return Type	Character
Overlapped?	No
Default	Stop Frequency of the previous segment. If first segment, start frequency of the analyzer.

SENSe<cnum>:SEGMent<snum>:FREQuency:STOP <num>

(Read-Write) Sets the Stop Frequency for the specified sweep segment.

Note: All other segment's Start and Stop Frequency values that are larger than this frequency are changed to this frequency.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<num>	Stop Frequency in Hz. Choose any number between the **minimum** and **maximum** frequency of the analyzer. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.

Examples	SENS:SEGM:FREQ:STOP 1MHZ sense2:segment2:frequency:stop maximum
Query Syntax	SENSe<cnum>:SEGMent<snum>:FREQuency:STOP?
Return Type	Character
Overlapped?	No
Default	If first segment, stop frequency of the analyzer. Otherwise, start frequency of the segment.

SENSe<cnum>:SEGMent<snum>:POWer[<port>][:LEVel] <num>

(Read-Write) Sets the Port Power level for the specified sweep segment.
First set SENS:SEGM:POW:CONTrol ON.

All subsequent segments that are added assume the new Power Level value.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to modify. Choose any existing segment number.
<port>	Port number of the source. Choose from 1 or 2. If unspecified, value is set to 1.
<num>	Power level. Choose from any number between: **-90** and **20**
Examples	SENS:SEGM:POW 0 sense2:segment2:power1:level -10
Query Syntax	SENSe<cnum>:SEGMent<snum>:POWer[<port>][:LEVel]?
Return Type	Character
Overlapped?	No
Default	0

SENSe<cnum>:SEGMent:POWer[:LEVel]:CONTrol <ON | OFF>

(Read-Write) Specifies whether Power Level can be set independently for each segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns Power Level control ON. Power level can be set for each segment. **OFF** (or 0) - turns Power Level control OFF. Use the channel power level setting.
Examples	SENS:SEGM:POW:CONT ON sense2:segment:power:level:control off
Query Syntax	SENSe<cnum>:SEGMent:POWer[:LEVel]:CONTrol?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

SENSe<cnum>:SEGMent<snum>[:STATe] <ON | OFF>

(Read-Write) Turns the specified sweep segment ON or OFF.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Segment number to be turned ON or OFF
<ON | OFF>	**ON** (or 1) - turns segment ON.

	OFF (or 0) - turns segment OFF.
Examples	SENS:SEGM ON sense2:segment2:state off
Query Syntax	SENSe<cnum>:SEGMent[:STATe]? <snum>
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

SENSe<cnum>:SEGMent<snum>:SWEep:POINts <num>

(Read-Write) Sets the number of data points for the specified sweep segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Any existing segment number. If unspecified, value is set to 1
<num>	Number of points in the segment. The total number of points in all segments cannot exceed **16001.** A segment can have as few as 1 point. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SEGM:SWE:POIN 51 sense2:segment2:sweep:points maximum
Query Syntax	SENSe<cnum>:SEGMent<snum>:SWEep:POINts?
Return Type	Character
Overlapped?	No
Default	201

SENSe<cnum>:SEGMent<snum>:SWEep:TIME <num>

(Read-Write) Sets the time the analyzer takes to sweep the specified sweep segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Any existing segment number.
<num>	Sweep time in seconds. Choose a number between **0** and **100** **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SEGM:SWE:TIME 1ms sense2:segment2:sweep:time .001
Query Syntax	SENSe<cnum>:SEGMent<snum>:SWEep:TIME?
Return Type	Character
Overlapped?	No
Default	Not Applicable

SENSe<cnum>:SEGMent:SWEep:TIME:CONTrol <ON | OFF>

(Read-Write) Specifies whether Sweep Time can be set independently for each sweep segment.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns Sweep Time control ON. Sweep Time can be set for each segment.

OFF (or 0) - turns Sweep Time control OFF. Uses the channel Sweep Time setting.

Examples	SENS:SEGM:SWE:TIM:CONT ON sense2:segment:sweep:time:control off
Query Syntax **Return Type**	SENSe<cnum>:SEGMent:SWEep:TIME:CONTrol? Boolean (1 = ON, 0 = OFF)
Overlapped? **Default**	No OFF

SENSe<cnum>:SEGMent<snum>:X:SPACing <char>

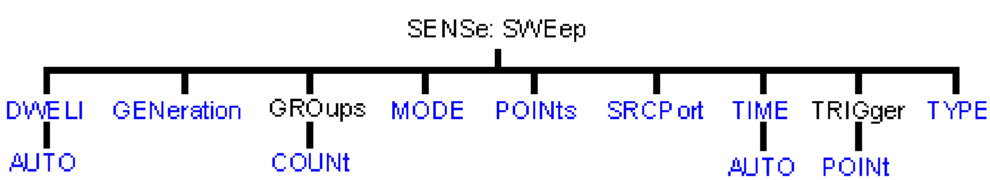
(Read-Write) Sets X-axis spacing ON or OFF

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<snum>	Any existing segment number. (This parameter is ignored)
<char>	**LINear** - turns X-axis point spacing OFF **OBASe** - turns X-axis point spacing ON
Examples	SENS:SEGM:X:SPACing LIN sense2:segment1:spacing obase
Query Syntax **Return Type**	SENSe<cnum>:SEGMent<snum>:X:SPACing? Character
Overlapped? **Default**	No LINear

Sense:Sweep Commands

Specifies the sweep functions of the analyzer.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Sweeping

SENSe<cnum>:SWEep:DWELl <num>

(Read-Write) Sets the dwell time between each sweep point.

- Dwell time is **ONLY** available with SENSe:SWEep:GENeration set to **STEPped;** It is **Not** available in **ANALOG**.

Sending dwell = 0 is the same as setting SENS:SWE:DWEL:AUTO **ON.** Sending a dwell time > 0 sets SENS:SWE:DWEL:AUTO **OFF**.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Dwell time in seconds. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SWE:DWEL .1 sense2:sweep:dwell min
Query Syntax	SENSe<cnum>:SWEep:DWELl?
Return Type	Character
Overlapped?	No
Default	0 - (**Note**: dwell time set to 0 is the same as dwell:auto ON)

SENSe<cnum>:SWEep:DWELl:AUTO <ON | OFF>

(Read-Write) Specifies whether or not to automatically calculate and set the minimum possible dwell time. Setting Auto **ON** has the same effect as setting dwell time to **0**.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns dwell ON. **OFF** (or 0) - turns dwell OFF.
Examples	SENS:SWE:DWEL:AUTO ON sense2:sweep:dwell:auto off
Query Syntax	SENSe<cnum>:SWEep:DWELl:AUTO?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

SENSe<cnum>:SWEep:GENeration <char>

(Read-Write) Sets sweep as Stepped or Analog.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: **STEPped** - source frequency is CONSTANT during measurement of eah displayed point. More accurate than ANALog. Dwell time can be set in this mode. **ANALog** - source frequency is continuously RAMPING during measurement of each displayed point. Faster than STEPped. Sweep

time (not dwell time) can be set in this mode.

Examples	SENS:SWE:GEN STEP sense2:sweep:generation analog
Query Syntax	SENSe<cnum>:SWEep:GENeration?
Return Type	Character
Overlapped?	No
Default	Analog

SENSe<cnum>:SWEep:GROups:COUNt <num>

(Read-Write) Sets the trigger count (groups) for the specified channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Count (groups) number. Choose any number between: **1** and **2e6** (1 is the same as single trigger)
Examples	SENS:SWE:GRO:COUN 10 sense2:sweep:groups:count 50
Query Syntax	SENSe<cnum>:SWEep:GROups:COUNt?
Return Type	Character
Overlapped?	No
Default	1

SENSe<cnum>:SWEep:MODE <char>

(Read-Write) Sets the trigger mode for the specified channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Trigger mode. Choose from: **HOLD** - channel will not trigger **CONTinuous** - channel triggers indefinitely **GROups** - channel accepts the number of triggers specified with the last SENS:SWE:GRO:COUN <num>
Examples	SENS:SWE:MODE CONT sense2:sweep:mode hold
Query Syntax	SENSe<cnum>:SWEep:MODE?
Return Type	Character
Overlapped?	**YES** - SENS:SWE:MODE GROUPS (when INIT:CONT is ON) **NO** - HOLD and CONTinuous
Default	CONTinuous

SENSe<cnum>:SWEep:POINts <num>

(Read-Write) Sets the number of data points for the measurement.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Choose any number between **1** and **16001** **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SWE:POIN 51 sense2:sweep:points max
Query Syntax	SENSe<cnum>:SWEep:POINts?
Return Type	Character
Overlapped?	No
Default	201

SENSe<cnum>:SWEep:SRCPort <1 | 2>

(Read-Write) Sets the source port when making non S-parameter measurements. Has no effect on S-parameter measurements.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<1 | 2>	**1** - Source power comes out Port 1 **2** - Source power comes out Port 2
Examples	SENS:SWE:SRCP 1 sense2:sweep:srcport 2
Query Syntax	SENSe<cnum>:SWEep:SRCPort?
Return Type	Character
Overlapped?	No
Default	1

SENSe<cnum>:SWEep:TIME <num>

(Read-Write) Sets the time the analyzer takes to complete one sweep.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Sweep time in seconds. Choose a number between **0** and **86,400** (24hrs) **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SENS:SWE:TIME 1ms sense2:sweep:time .001
Query Syntax	SENSe<cnum>:SWEep:TIME?

Return Type	Character
Overlapped?	No
Default	NA

SENSe<cnum>:SWEep:TIME:AUTO <ON | OFF>

(Read-Write) Turns the automatic sweep time function ON or OFF.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns the automatic sweep time ON. **OFF** (or 0) - turns the automatic sweep time OFF.
Examples	SENS:SWE:TIME:AUTO sense2:sweep:time:auto off
Query Syntax	SENSe<cnum>:SWEep:TIME:AUTO?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

SENSe<cnum>:SWEep:TRIGger:POINt <ON | OFF>

(Read-Write) Specifies whether the specified channel will measure one point for each trigger or all of the measurements in the channel. Setting any channel to POINt mode will automatically set the TRIGger:SCOPe = CURRent.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - Channel measures one data point per trigger. **OFF** (or 0) - All measurements in the channel made per trigger.
Examples	SENS:SWE:TRIG:POIN ON sense2:sweep:trigger:point off
Query Syntax	SENSe<cnum>:SWEep:TRIGger:POINt?
Return Type	Boolean (1 = Point, 0 = Measurement)
Overlapped?	No
Default	0 - Measurement

SENSe<cnum>:SWEep:TYPE <char>

(Read-Write) Sets the type of analyzer sweep mode.

Parameters

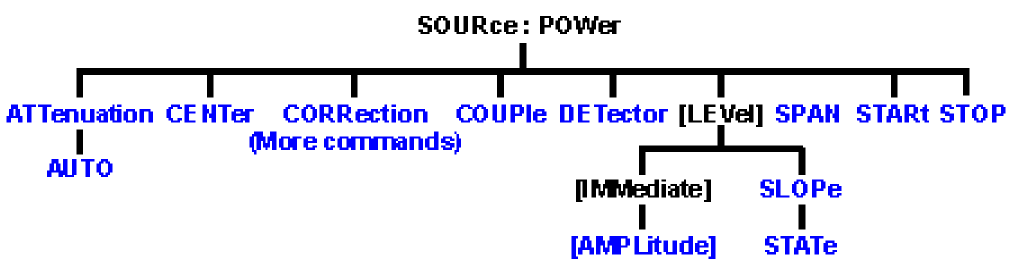
<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: **LINear | LOGarithmic | POWer | CW | SEGMent** **Note**: SWEep TYPE cannot be set to SEGMent if there are no segments turned ON. A segment is automatically turned ON when

the analyzer is started.

Examples	SENS:SWE:TYPE LIN sense2:sweep:type segment
Query Syntax	SENSe<cnum>:SWEep:TYPE?
Return Type	Character
Overlapped?	No
Default	LINear

Source Commands

Controls the power delivered to the DUT.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Power Settings

SOURce<cnum>:POWer<port>:ATTenuation <num>

(Read-Write) Sets the attenuation level for the selected channel. Sending this command turns automatic attenuation control (SOUR:POW:ATT:AUTO) to OFF. If the ports are coupled, changing the attenuation on one port will also change the attenuation on the other port. To turn port coupling OFF use SOURce:POWer:COUPle OFF.

Note: Attenuation cannot be set with **Sweep Type** set to **Power**

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number of the attenuator being set. Choose **1** or **2**; If unspecified, value is set to 1.
<num>	Choose a number between **0** and **70** dB, in 10 dB steps. If a number other than these is entered, the analyzer will select the next lower valid value. For example, if 19.9 is entered for <num> the analyzer will switch in 10 dB attenuation. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SOUR:POW:ATT 10 source2:power:attentuation maximum

Query Syntax	SOURce<cnum>:POWer<port>:ATTenuation?
Return Type	Character
Overlapped?	No
Default	0

SOURce<cnum>:POWer<port>:ATTenuation:AUTO <ON | OFF>

(Read-Write) Turns automatic attenuation control ON or OFF. Setting an attenuation value (using SOURce:POWer:ATTenuation <num>) sets AUTO **OFF**.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1.
<port>	Port number of the attenuator being set. Choose **1** or **2**; If unspecified, value is set to 1.
<ON | OFF>	**ON** (or 1) - turns coupling ON. The analyzer automatically selects the appropriate attenuation level to meet the specified power level. **OFF** (or 0) - turns coupling OFF. Attenuation level must be set using SOURce:POWer:ATTenuation <num>.

Examples	SOUR:POW2:ATT:Auto On source2:power: attentuation:auto off
Query Syntax	SOURce<cnum>:POWer:ATTenuation:Auto?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

SOURce<cnum>:POWer:CENTer <num>

(Read-Write) Sets the power sweep center power. Must also set:
SENS:SWE:TYPE POWer and SOURce:POWer:SPAN <num>**.**

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Center power. Choose a number between **-90** and **20** dBm (actual achievable leveled power depends on frequency)

Examples	SOUR:POW:CENT -15 source2:power:center -7
Query Syntax	SOURce<cnum>:POWer:CENTer?
Return Type	Character
Overlapped?	No
Default	0 dBm

SOURce<cnum>:POWer:COUPle <ON | OFF>

(Read-Write) Turns Port Power Coupling ON or OFF.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON | OFF>	**ON** (or 1) - turns coupling ON. Power level can be set individually for each source port. **OFF** (or 0) - turns coupling OFF. The same power level is used for both source ports.

Examples	SOUR:POW:COUP ON

source2:power:couple off

Query Syntax	SOURce<cnum>:POWer:COUPle?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	ON

SOURce<cnum>:POWer:DETector <INTernal | EXTernal>

(Read-Write) Sets the source leveling loop as Internal or External.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<INTernal | EXTernal>	**INTernal** - Internal leveling is applied to the source **EXTernal** - External leveling is applied to the source through a rear-panel jack.
Examples	SOUR:POW:DET INT source2:power:detector external
Query Syntax	SOURce<cnum>:POWer:DETector?
Return Type	Character
Overlapped?	No
Default	INTernal

SOURce<cnum>:POWer<port>[:LEVel][:IMMediate][:AMPLitude] <num>

(Read-Write) Sets the RF power output level.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number of the attenuator being set. Choose **1** or **2**; If unspecified, value is set to 1.
<num>	Source power in dBm. Choose any value between **-90** and **+20** dBm Actual achievable leveled power depends on frequency. **Note**: This command will accept **MIN** or **MAX** instead of a numeric parameter. See SCPI Syntax for more information.
Examples	SOUR:POW1 5DB source2:power:level :immediate:amplitude maximum
Query Syntax	SOURce<cnum>:POWer[:LEVel][:IMMediate][:AMPLitude]?
Return Type	Character
Overlapped?	No
Default	0 dBm

SOURce<cnum>:POWer[:LEVel]:SLOPe <int>

(Read-Write) Sets the RF power slope value.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<int>	Slope value in db/GHz. Choose any integer between **-2** and **2** (0 is no slope).
Examples	SOUR:POW:SLOP 2

source2:power:slope -2

Query Syntax	SOURce<cnum>:POWer[:LEVel]:SLOPe?
Return Type	Character
Overlapped?	No
Default	0

SOURce<cnum>:POWer[:LEVel]:SLOPe:STATe <ON|OFF>

(Read-Write) Turns Power Slope ON or OFF.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON|OFF>	**ON** (or 1) - turns slope ON. **OFF** (or 0) - turns slope OFF.
Examples	SOUR:POW:SLOP:STAT ON source2:power:slope:state off
Query Syntax	SOURce<cnum>:POWer[:LEVel]:SLOPe:STATe?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF

SOURce<cnum>:POWer:SPAN <num>

(Read-Write) Sets the power sweep span power. Must also set:

SENS:SWE:TYPE POWer and SOURce:POWer:CENTer <num>**.**

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Span power. Choose a number between: **-90** and **20** dBm (actual achievable leveled power depends on frequency)
Examples	SOUR:POW:SPAN -15 source2:power:span -7
Query Syntax	SOURce<cnum>:POWer:SPAN?
Return Type	Character
Overlapped?	No
Default	0 dBm

SOURce<cnum>:POWer:STARt <num>

(Read-Write) Sets the power sweep start power. Must also set SENS:SWE:TYPE POWer and SOURce:POWer:STOP <num>**.**

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Start power. Choose a number between **-90** and **+20** dBm (actual achievable leveled power depends on frequency)
Examples	SOUR:POW:STAR -15 source2:power:start -7
Query Syntax	SOURce<cnum>:POWer:STARt?
Return Type	Character

Overlapped?	No
Default	0 dBm

SOURce<cnum>:POWer:STOP <num>

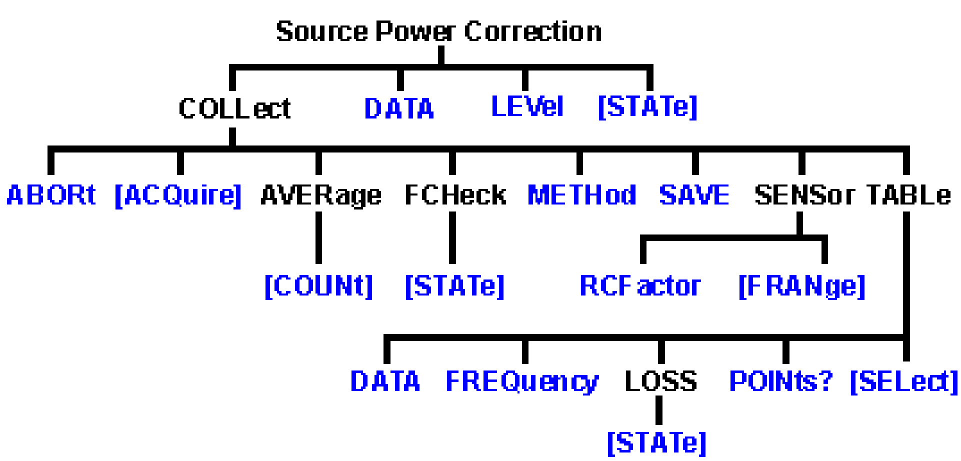
(Read-Write) Sets the power sweep stop power. Must also set:
SENS:SWE:TYPE POWer and SOURce:POWer:START <num>**.**

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<num>	Stop power. Choose a number between **-90** and **+20** dBm (actual achievable leveled power depends on frequency)
Examples	SOUR:POW:STOP -15 source2:power:stop -7
Query Syntax	SOURce<cnum>:POWer:STOP?
Return Type	Character
Overlapped?	No
Default	0 dBm

Source:Power:Correction Commands

Controls the source power correction features of the analyzer.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See an example program using these commands.
- See a template for creating your own Power Meter Driver
- Learn about Source Cal

Note: the SOURce:POWer:CORRection:COLLect:ACQuire command, used to step the PNA and read a power meter, cannot be sent over the GPIB. Use one of the alternative methods described in the command details.

SOURce<cnum>:POWer<port>:CORRection:COLLect:ABORt

(Write-only) Aborts a source power calibration sweep that is in progress.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
Examples	SOUR:POW:CORR:COLL:ABOR source1:power2:correction:collect:abort
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer<port>:CORRection:COLLect[:ACQuire] <char>

(Write-only) Initiates a source power cal acquisition sweep using the power sensor attached to the specified channel (A or B) on the power meter.

Note: Never use GPIB to send this SCPI command to the PNA. This command requires the PNA to take GPIB control. The PNA currently does not support pass control a technique whereby GPIB control can be passed back and forth between two controllers.

Use one of the following methods to perform this command or its equivalent:

- SCPI programming of the PNA using a LAN Client interface (see example)
- Send SCPI commands through the COM interface using the SCPI String Parser object.

Directly control the Power Meter and PNA to step frequency; then acquire and store the Power reading. (see example)

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<char>	Choose from: **ASENsor** - Sensor on power meter channel A **BSENsor** - Sensor on power meter channel B
Examples	SOUR:POW:CORR:COLL ASEN source1:power2:correction:collect:acquire bsensor
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer<port>:CORRection:COLLect:AVERage[:COUNt] <num>

(Read-Write) Specifies how many power readings are taken at each frequency point (averaging factor) during a source power cal acquisition sweep.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<num>	Number of readings per point. Choose any number between 1 and 100.

Examples	SOUR:POW:CORR:COLL:AVER 2 source1:power2:correction:collect:average:count 3
Query Syntax	SOURce:POWer:CORRection:COLLect:AVERage[:COUNt]?
Return Type	Character
Overlapped?	No
Default	1

SOURce<cnum>:POWer:CORRection:COLLect:FCHeck[:STATe] <ON | OFF>

(Read-Write) Enables and disables frequency checking of source power cal acquisition sweeps.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON|OFF>	**ON (1)** turns source power cal frequency checking ON. A requested acquisition will only succeed for those frequency points which fall within a frequency range specified for the power sensor being used. An acquisition will pause in mid-sweep if the frequency is about to exceed the maximum frequency limit specified for that sensor. When the sweep is paused in this manner, a sensor connected to the other channel input of the power meter can be connected to the measurement port in place of the previous sensor, and used to complete the sweep. However, the maximum frequency specified for the second sensor would need to be sufficient for the sweep to complete. Frequency limits are specified using the commands SOURce<cnum>:POWer:CORRection:COLLect:ASENsor[:FRANge] and SOURce<cnum>:POWer:CORRection:COLLect:BSENsor[:FRANge]. **OFF (0)** - turns source power cal frequency checking OFF. An acquisition will use just one power sensor for the entire sweep, regardless of frequency.

Examples	SOUR:POW:CORR:COLL:FCH ON source1:power2:correction:collect:fcheck:state off
Query Syntax	SOURce:POWer:CORRection:COLLect:FCHeck[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF (0)

SOURce<cnum>:POWer<port>:CORRection:COLLect:METHod <char>

(Read-Write) Selects the source power calibration method. Currently, PMETer is the only supported method. In general, test software should not omit use of this command as it may eventually be required if other source power cal methods become supported.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<char>	Choose from: **NONE** - No Cal method **PMETer** - Power Meter

Examples	SOUR:POW:CORR:COLL:METH PMET source1:power2:correction:collect:method pmeter
Query Syntax	SOURce:POWer:CORRection:COLLect:METHod?
Return Type	Character
Overlapped?	No
Default	NONE

SOURce<cnum>:POWer<port>:CORRection:COLLect:SAVE

(Write-only) Applies the array of correction values after a source power calibration sweep has completed. The source power correction will then be active on the specified source port for channel <cnum>. This command does NOT save the correction values.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
Examples	SOUR:POW:CORR:COLL:SAVE source1:power2:correction:collect:save
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer:CORRection:COLLect:<pmChan>SENsor[:FRANge] <num1>,<num2>

(Read-Write) Specifies the frequency range over which the power sensors connected to the specified channels (A and B) of the power meter can be used (minimum frequency, maximum frequency). If the power meter has only a single channel, that channel is considered channel A.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pmChan>	Power Meter channel. Choose from: **A** - Channel A **B -** Channel B
<num1>	Minimum frequency for the sensor. If a frequency unit is not specified, Hz is assumed. No limits are placed on this value.
<num2>	Maximum frequency for the sensor. If a frequency unit is not specified, Hz is assumed. No limits are placed on this value.
Examples	SOUR:POW:CORR:COLL:ASEN 100E3, 3E9 source1:power2:correction:collect:bsensor:frange 10 MHz, 18 GHz
Query Syntax	SOURce:POWer:CORRection:COLLect:ASENsor[:FRANge]? SOURce:POWer:CORRection:COLLect:BSENsor[:FRANge]?
Return Type	Character
Overlapped?	No
Default	0,0

SOURce<cnum>:POWer:CORRection:COLLect:<pmChan>SENsor:RCFactor <num>

(Read-Write)) Specifies the reference cal factor for the power sensor connected to channel A or B of the power meter. If the power meter has only a single channel, that channel is considered channel A.

Note: If the sensor connected to the specified channel of the power meter contains cal factors in EPROM (such as the Agilent E-series power sensors), those will be the cal factors used during the calibration sweep. The reference cal factor value associated with this command, and any cal factors entered into the PNA for that sensor channel, will not be used.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<pmChan>	Power Meter channel. Choose from: **A** - Channel A **B -** Channel B
<num>	Reference cal factor in percent. Choose any number between 1 and 150.
Examples	SOUR:POW:CORR:COLL:ASEN:RCF 98.7 source1:power2:correction:collect:bsensor:rcfactor 105
Query Syntax	SOURce:POWer:CORRection:COLLect:ASENsor:RCFactor? SOURce:POWer:CORRection:COLLect:BSENsor:RCFactor?
Return Type	Character
Overlapped?	No
Default	100

SOURce<cnum>:POWer:CORRection:COLLect:TABLe:DATA <data>

(Read-Write) Read or write data into the selected table. If the selected table is a power sensor table, the data is interpreted as cal factors in units of percent. If the loss table is selected, the data is interpreted as loss in units of dB.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<data>	Data to write into the selected table.
Examples	SOURce:POWer:CORRection:COLLect:TABLe:DATA 0.12, 0.34, 0.56
Query Syntax	SOURce<cnum>:POWer:CORRection:COLLect:TABLe:DATA? If the selected table is currently empty, no data is returned.
Return Type	Character - one number per table segment
Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer:CORRection:COLLect:TABLe:FREQuency <data>

(Read-Write) Read or write frequency values for the selected table (cal factor table for a power sensor, or the loss compensation table).

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<data>	Frequency data to write into the selected table.
Examples	SOURce:POWer:CORRection:COLLect:TABLe:FREQuency 10E6, 1.5E9, 9E9
Query Syntax	SOURce<cnum>:POWer:CORRection:COLLect:TABLe:FREQuency? If the selected table is currently empty, no data is returned.
Return Type	Character - one number per table segment

Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer:CORRection:COLLect:TABLe:LOSS[:STATe] <ON | OFF>

(Read-Write) Indicates whether or not to adjust the power readings using the values in the loss table during a source power cal sweep.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<ON|OFF>	**ON (or 1) -** turns use of the loss table ON. **OFF (or 0)** - turns use of the loss table OFF.
Examples	SOUR:POW:CORR:COLL:TABL:LOSS ON source1:power2:correction:collect:table:loss:state off
Query Syntax	SOURce:POWer:CORRection:COLLect:TABLe:LOSS[:STATe]?
Return Type	Boolean (1 = ON, 0 = OFF)
Overlapped?	No
Default	OFF (0)

SOURce<cnum>:POWer:CORRection:COLLect:TABLe:POINts?

(Read-only) Returns the number of segments that are currently in the selected table.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
Examples	SOUR:POW:CORR:COLL:TABL:POIN? source1:power2:correction:collect:table:points?
Return Type	Character
Overlapped?	No
Default	0

SOURce<cnum>:POWer:CORRection:COLLect:TABLe[:SELect] <char>

(Read-Write) Selects which table (cal factor table for a power sensor, or the loss compensation table) you want to write to or read from. Read or write using SOURce:POWer:CORRection:COLLect:TABLe:FREQuency and SOURce:POWer:CORRection:COLLect:TABLe:DATA

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: **NONE** - No table selected **ASENsor** - Cal Factor table for Power Sensor A **BSENsor** - Cal Factor table for Power Sensor B **LOSS** - Loss compensation table
Examples	SOUR:POW:CORR:COLL:TABL ASEN source1:power2:correction:collect:table:select bsensor
Query Syntax	SOURce:POWer:CORRection:COLLect:TABLe[:SELect]?
Return Type	Character

Overlapped?	No
Default	NONE

SOURce<cnum>:POWer<port>:CORRection:DATA <data>

(Read-Write) Writes and reads source power calibration data.

When querying source power calibration data, if no source power cal data exists for the specified channel and source port, no data is returned.

If a change in the instrument state causes interpolation and/or extrapolation of the source power cal, the correction data associated with this command correspond to the new instrument state (interpolated and/or extrapolated data).

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<data>	Correction Data
Examples	SOURce1:POWer2:CORRection:DATA 0.12, -0.34, 0.56
Query Syntax	SOURce<cnum>:POWer<port>:CORRection:DATA?
Return Type	Character - One number per trace point
Overlapped?	No
Default	Not Applicable

SOURce<cnum>:POWer<port>:CORRection:LEVel <num>

(Read-Write) Specifies the power level that is expected at the desired reference plane (DUT input or output).

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<num>	Cal power level in dBm. Because this could potentially be at the output of a device-under-test, no limits are placed on this value here. It is realistically limited by the specifications of the device (power sensor) that will be used for measuring the power. The power delivered to the PNA receiver must never exceed PNA specifications for the receiver!
Examples	SOUR:POW:CORR:LEV 10 source1:power2:correction:level 0 dbm
Query Syntax	SOURce:POWer:CORRection:LEVel?
Return Type	Character
Overlapped?	No
Default	0 dBm

SOURce<cnum>:POWer<port>:CORRection[:STATe] <ON|OFF>

(Read-Write) Enables and disables source power correction for the specified port on the specified channel.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1
<port>	Port number to correct for source power. If unspecified, value is set to 1.
<ON|OFF>	ON (or 1) turns source power correction ON. OFF (or 0) - turns source power correction OFF.
Examples	SOUR:POW:CORR ON

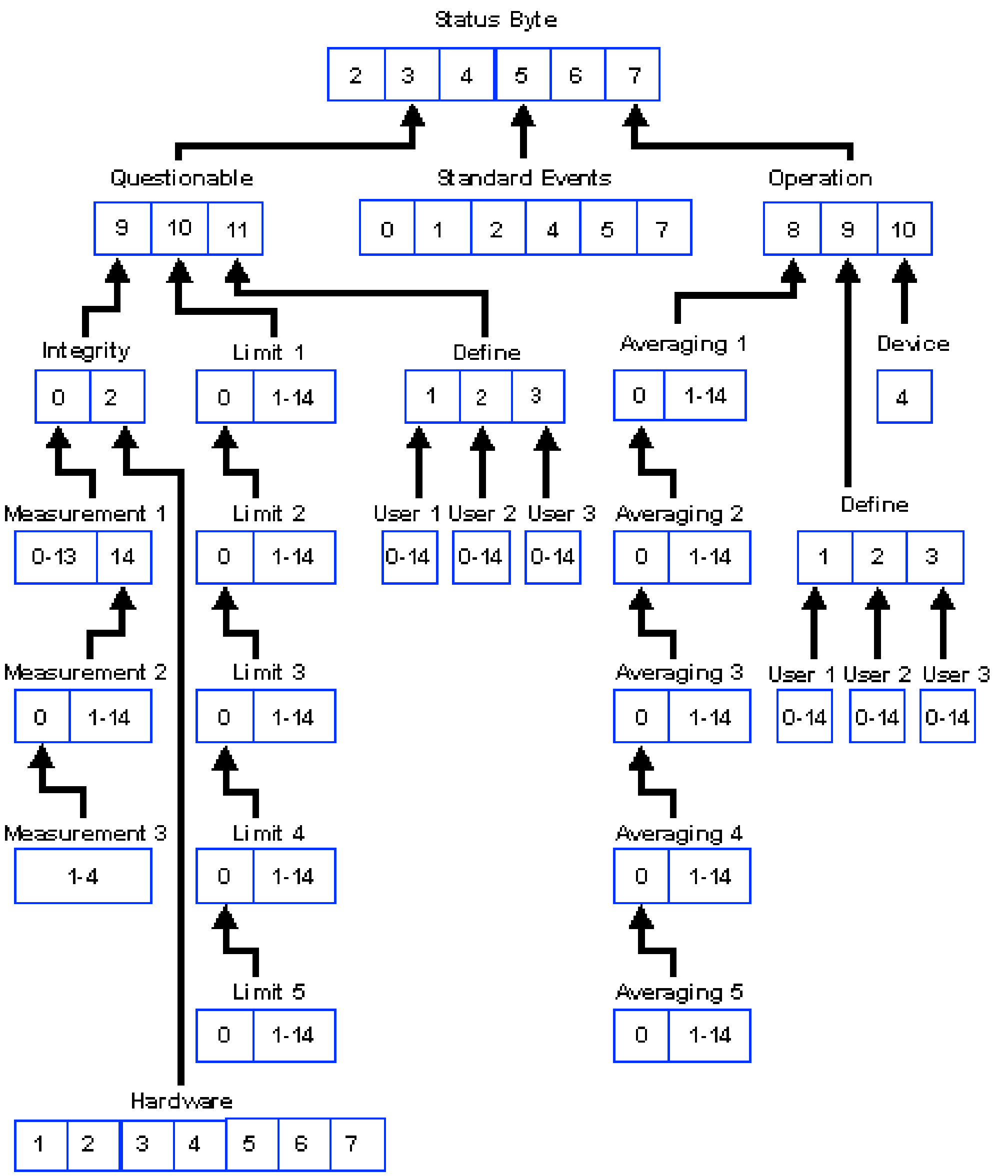
source1:power2:correction:state off

Query Syntax SOURce:POWer:CORRection[:STATe]?
Return Type Boolean (1 = ON, 0 = OFF)

Overlapped? No
Default OFF (0)

Status Register Commands

The status registers enable you to query the state of selected events that occur in the analyzer.



Note: This documentation requires familiarity with the "Standard Status Data Structure - Register Model" as defined in IEEE Std 488.2-1992.

- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Status Registers

Note: Any bit not shown in the registers is not used but may be reserved for future use.

Status Byte Register
Summarizes the states of the other registers and monitors the analyzer's output queue. It also generates **service requests**. The Enable register is called the Service Request Enable Register.

Commands	**Description**
*CLS	Clears ALL "event" registers and the SCPI Error / Event queue. The corresponding ENABLE registers are unaffected.
*STB?	Reads the value of the analyzer's status byte. The byte remains after being read.

*SRE?	Reads the current state of the Service Request **Enable** Register.
*SRE *<num>*	Sets bits in the Service Request **Enable** register. The current setting of the SRE register is stored in non-volatile memory. Use *SRE 0 to clear the enable.
	<num> Combined value of the weights for bits to be set.

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
2	4	Error / Event queue Summary (EAV)	the Error / Event queue is not empty. To read the the error message, use SYST:ERR?
3	8	Questionable Register Summary	any enabled bit in the **questionable** event status register is set to 1
4	16	Message Available	the output queue is not empty
5	32	Standard Event Register Summary	any enabled bit in the **standard** event status register is set to 1
6	64	Request Service	any of the other bits in the status byte register is set to 1 (used to alert the controller of a service request within the analyzer). This bit cannot be disabled.
7	128	Operation Register Summary	any enabled bit in the standard **operation** event status register is set to 1

STATus:QUEStionable:<keyword>

Summarizes conditions that monitor the quality of measurement data.

<keyword>	Example
:CONDition?	STAT:QUES:COND?
:ENABle <bits>	STAT:QUES:ENAB 1024
[:EVENt]?	STAT:QUES?
:NTRansition <bits>	STAT:QUES:NTR 1024
:PTRansition <bits>	STAT:QUES:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
9	512	Integrity Reg summary	any enabled bit in the **Integrity** event register is set to 1
10	1024	Limit Registers summary	any enabled bit in the **Limit** event registers is set to 1
11	2048	Define Registers summary	any enabled bit in the **Define** event registers is set to 1

STATus:QUEStionable:INTegrity <keyword>

Summarizes conditions in the Measurement Integrity register.

<keyword>	Example
:CONDition?	STAT:QUES:INT:COND?
:ENABle <bits>	STAT:QUES:INT:ENAB 1024
[:EVENt]?	STAT:QUES:INT?
:NTRansition <bits>	STAT:QUES:INT:NTR 1024
:PTRansition <bits>	STAT:QUES:INT:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Measurement Summary	any bit in the **Measurement Integrity** event register is set to 1
2	2	Hardware Summary	any bit in the **Hardware** event register is set to 1

STATus:QUEStionable:INTegrity:HARDware<keyword>

Monitors the status of hardware failures.

<keyword>	**Example**
:CONDition?	STAT:QUES:INT:HARD:COND?
:ENABle <bits>	STAT:QUES:INT:HARD:ENAB 1024
[:EVENt]?	STAT:QUES:INT:HARD?
:NTRansition <bits>	STAT:QUES:INT:HARD:NTR 1024
:PTRansition <bits>	STAT:QUES:INT:HARD:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
1	2	Phase Unlock	the source has lost phaselock, possibly caused by a reference channel open or a hardware failure.
2	4	Unleveled	the source power is unleveled. This could be caused by a source set for more power than it can deliver at the tuned frequency. Or it could be caused by a hardware failure.
3	8	Overpower	too much power is detected at the input. This is from either using an amplifier, or a hardware failure.
4	16	EE Write Failed	an attempted write to the EEPROM has failed, possibly caused by a hardware failure.
5	32	YIG Cal Failed	the analyzer was unable to calibrate the YIG. Either the phaselock has been lost or there has been a hardware failure.
6	64	Ramp Cal Failed	the analyzer was unable to calibrate the analog ramp generator due to a possible hardware failure.
7	128	OverTemp	the source temperature sensor exceeds the limit. It could result from restricted airflow or a broken fan

STATus:QUEStionable:INTegrity:MEASurement<n> <keyword>

Monitors the lag between changing a channel settings and when the data is ready to query out.

When you change the channel state (start/stop freq, bandwidth, and so on), then the questionable bit for that channel gets set. This indicates that your desired channel state does not yet match the data you would get if querying a data trace. When the next complete sweep has been taken (without aborting in the middle), and the data trace matches the channel state that produced it, the bit is cleared for that channel.

<n>	Measurement register number. Choose from 1 to 3
<keyword>	**Example**
:CONDition?	STAT:QUES:INT:MEAS1:COND?
:ENABle <bits>	STAT:QUES:INT:MEAS2:ENAB 1024
[:EVENt]?	STAT:QUES:INT:MEAS3?
:NTRansition <bits>	STAT:QUES:INT:MEAS2:NTR 1024
:PTRansition <bits>	STAT:QUES:INT:MEAS1:PTR 0

		Measurement Register <n>			
Bit	**Weight**	**1**	**2**	**3**	**Bit is set to 1 when the following conditions exist:**
0	1	1	Summary from Meas Reg 3		a setting change on this channel has occurred and the data does not yet reflect that change.
1	2	2	15	29	a setting change on this channel has occurred and the data does not yet reflect that change.

2	4	3	16	30	a setting change on this channel has occurred and the data does not yet reflect that change.
3	8	4	17	31	a setting change on this channel has occurred and the data does not yet reflect that change.
4	16	5	18	32	a setting change on this channel has occurred and the data does not yet reflect that change.
5	32	6	19		a setting change on this channel has occurred and the data does not yet reflect that change.
6	64	7	20		a setting change on this channel has occurred and the data does not yet reflect that change.
7	128	8	21		a setting change on this channel has occurred and the data does not yet reflect that change.
8	256	9	22		a setting change on this channel has occurred and the data does not yet reflect that change.
9	512	10	23		a setting change on this channel has occurred and the data does not yet reflect that change.
10	1024	11	24		a setting change on this channel has occurred and the data does not yet reflect that change.
11	2048	12	25		a setting change on this channel has occurred and the data does not yet reflect that change.
12	4096	13	26		a setting change on this channel has occurred and the data does not yet reflect that change.
13	8192	14	27		a setting change on this channel has occurred and the data does not yet reflect that change.
14	16384	Summary from Meas Reg 2	28		a setting change on this channel has occurred and the data does not yet reflect that change.

STATus:QUEStionable:LIMit<n> <keyword>

Monitors and summarizes the status of limit line failures. When a trace fails, the representative bit is set to 1. Bit 0 is used to summarize failures in the registers that follow. For example, Limit 3 register, bit 0, summarizes the failures from registers 4 and 5.

All enable bits are set to 1 by default. To find the measurement number, use Calc:Par:Mnum

<n>	Limit register: Choose from 1 to 5.
<keyword>	**Example**
:CONDition?	STAT:QUES:LIM4:COND?
:ENABle <bits>	STAT:QUES:LIM1:ENAB 1024
[:EVENt]?	STAT:QUES:LIM3?
:NTRansition <bits>	STAT:QUES:LIM2:NTR 1024
:NTRansition?	STAT:QUES:LIM1:NTR?
:PTRansition <bits>	STAT:QUES:LIM5:PTR 0
:PTRansition?	STAT:QUES:LIM1:PTR?

		Limit Register <n>					
Bit	**Weight**	**1**	**2**	**3**	**4**	**5**	**Bit is set to 1 when the following conditions exist:**
0	1	2, 3, 4, 5	3, 4, 5	4, 5	5		Summary - Any point from these registers fails
		Trace Numbers					
1	2	1	15	29	43	57	any point on trace fails the limit test
2	4	2	16	30	44	58	any point on trace fails the limit test
3	8	3	17	31	45	59	any point on trace fails the limit test

4	16	4	18	32	46	60	any point on trace fails the limit test
5	32	5	19	33	47	61	any point on trace fails the limit test
6	64	6	20	34	48	62	any point on trace fails the limit test
7	128	7	21	35	49	63	any point on trace fails the limit test
8	256	8	22	36	50	64	any point on trace fails the limit test
9	512	9	23	37	51		any point on trace fails the limit test
10	1024	10	24	38	52		any point on trace fails the limit test
11	2048	11	25	39	53		any point on trace fails the limit test
12	4096	12	26	40	54		any point on trace fails the limit test
13	8192	13	27	41	55		any point on trace fails the limit test
14	16384	14	28	42	56		any point on trace fails the limit test

STATus:QUEStionable:DEFine<keyword>

Summarizes conditions in the Questionable:Define:User<1|2|3> event registers.

<keyword>	**Example**
:CONDition?	STAT:QUES:DEF:COND?
:ENABle <bits>	STAT:QUES:DEF:ENAB 1024
[:EVENt]?	STAT:QUES:DEF?
:NTRansition <bits>	STAT:QUES:DEF:NTR 1024
:PTRansition <bits>	STAT:QUES:DEF:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
1	2	USER1	any bit in the **USER1** event register is set to 1
2	4	USER2	any bit in the **USER2** event register is set to 1
3	8	USER3	any bit in the **USER3** event register is set to 1

STATus:QUEStionable:DEFine:USER<1|2|3><keyword>

Monitors conditions that you define and map in any of the three QUES:DEF:USER event registers.

<keyword>	**Example**
:ENABle <bits>	STAT:QUES:DEF:USER1:ENABle 1024
[:EVENt]?	STAT:QUES:DEF:USER1?
:MAP <bit>,<error>	STAT:QUES:DEF:USER1:MAP 0,-113 'when error -113 occurs, bit 0 in USER1 will set to 1.

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	for user	user defined
1	2	for user	user defined
2	4	for user	user defined
3	8	for user	user defined
4	16	for user	user defined
5	32	for user	user defined
6	64	for user	user defined
7	128	for user	user defined
8	256	for user	user defined
9	512	for user	user defined
10	1024	for user	user defined
11	2048	for user	user defined
12	4096	for user	user defined
13	8192	for user	user defined
14	16384	for user	user defined

Standard Event Status Register

Monitors "standard" events that occur in the analyzer. This register can only be cleared by:

- a Clear Command (*CLS).
- reading the Standard Enable Status Register (*ESE?).

a power-on transition. The analyzer clears the register and then records any transitions that occur, including setting the Power On bit (7).

Commands	Description
*ESE?	Reads the settings of the standard event **ENABLE** register.
*ESE ***<bits>***	Sets bits in the standard event **ENABLE** register. The current setting is saved in non-volatile memory. ***<bits>*** The sum of weighted bits in the register. Use *ESE 0 to clear the enable register.
*ESR?	Reads and clears the **EVENT** settings in the Standard Event Status register.
*OPC	Sets bit 0 when the overlapped command is complete. (see Understanding Command Synchronization / OPC).
*OPC?	Operation complete query - read the Operation Complete bit (0).

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Operation Complete	the two following events occur **in order:** 1. the *OPC command is sent to the analyzer the analyzer completes all pending overlapped commands
1	NA	Request Control	Not Supported - the analyzer application is not configured to control GPIB operation
2	4	Query Error	a query error is detected indicating: - an attempt to read data from the output queue when no data was present **OR** - data in the output queue was lost, as in an overflow
4	16	Execution Error	an execution error is detected indicating: - a <PROGRAM DATA> element was outside the legal range or inconsistent with the operation of the analyzer **OR** **-** the analyzer could not execute a valid command due to some internal condition
5	32	Command Error	a command error is detected indicating that the analyzer received a command that: • did not follow proper syntax • was misspelled was an optional command it does not implement
7	128	Power ON	Power to the analyzer has been turned OFF and then ON since the last time this register was read.

STATus:OPERation<keyword>

Summarizes conditions in the Averaging and Operation:Define:User<1|2|3> event registers.

<keyword>	**Example**
:CONDition?	STAT:OPER:COND?
:ENABle <bits>	STAT:OPER:ENAB 1024
[:EVENt]?	STAT:OPER?
:NTRansition <bits>	STAT:OPER:NTR 1024
:PTRansition <bits>	STAT:OPER:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
8	256	Averaging summary	either enabled bit in the **Averaging summary** event register is set to 1

9	512	User Defined summary	
10	1024	Device summary	either enabled bit in the **Device summary** event register is set to 1

STATus:OPERation:AVERaging<n> <keyword>

Monitors and summarizes the status of Averaging on traces 1 to 64. When averaging for a trace is complete, the representative bit is set to 1. Bit 0 is used to summarize the status in the registers that follow. For example, Average 3 register, bit 0, summarizes the status from registers 4 and 5. All enable bits are set to 1 by default. To find the measurement number, use Calc:Par:Mnum.

<n>	Averaging Register. Choose from 1 to 5
<keyword>	**Example**
:CONDition?	STAT:OPER:AVER1:COND?
:ENABle <bits>	STAT:OPER:AVER1:ENAB 1024
[:EVENt]?	STAT:OPER:AVER1?
:NTRansition <bits>	STAT:OPER:AVER1:NTR 1024
:PTRansition <bits>	STAT:OPER:AVER1:PTR 0

		Averaging Register <n>					
Bit	**Weight**	**1**	**2**	**3**	**4**	**5**	**Bit is set to 1 when the following conditions exist:**
0	1	2, 3, 4, 5	3, 4, 5	4, 5	5		any enabled bit in these registers is set to 1(Summary Bit)
		Trace Numbers					
1	2	1	15	29	43	57	Averaging on this trace is complete
2	4	2	16	30	44	58	Averaging on this trace is complete
3	8	3	17	31	45	59	Averaging on this trace is complete
4	16	4	18	32	46	60	Averaging on this trace is complete
5	32	5	19	33	47	61	Averaging on this trace is complete
6	64	6	20	34	48	62	Averaging on this trace is complete
7	128	7	21	35	49	63	Averaging on this trace is complete
8	256	8	22	36	50	64	Averaging on this trace is complete
9	512	9	23	37	51		Averaging on this trace is complete
10	1024	10	24	38	52		Averaging on this trace is complete
11	2048	11	25	39	53		Averaging on this trace is complete
12	4096	12	26	40	54		Averaging on this trace is complete
13	8192	13	27	41	55		Averaging on this trace is complete
14	16384	14	28	42	56		Averaging on this trace is complete

STATus:OPERation:DEFine<keyword>

Summarizes conditions in the OPERation:Define:User<1|2|3> event registers.

<keyword>	**Example**
:CONDition?	STAT:OPER:DEF:COND?
:ENABle <bits>	STAT:OPER:DEF:ENAB 12
[:EVENt]?	STAT:OPER:DEF?
:NTRansition <bits>	STAT:OPER:DEF:NTR 12
:PTRansition <bits>	STAT:OPER:DEF:PTR 0

Bit	**Weight**	**Description**	**Bit is set to 1 when the following conditions exist:**
1	2	USER1	any bit in the **USER1** event register is set to 1
2	4	USER2	any bit in the **USER2** event register is set to 1

3	8	USER3	any bit in the **USER3** event register is set to 1

STATus:OPERation:DEFine:USER<1|2|3><keyword>

Monitors conditions that you define and map in any of the three OPER:DEF:USER event registers.

<keyword>	**Example**
:ENABle <bits>	STAT:OPER:DEF:USER1:ENAB 1024
[:EVENt]?	STAT:OPER:DEF:USER1?
:MAP <bit>,<error>	STAT:OPER:DEF:USER1:MAP 0,-113 'when error -113 occurs, bit 0 in USER1 will set to 1.

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	for user	user defined
1	2	for user	user defined
2	4	for user	user defined
3	8	for user	user defined
4	16	for user	user defined
5	32	for user	user defined
6	64	for user	user defined
7	128	for user	user defined
8	256	for user	user defined
9	512	for user	user defined
10	1024	for user	user defined
11	2048	for user	user defined
12	4096	for user	user defined
13	8192	for user	user defined
14	16384	for user	user defined

STATus:OPERation:DEVice<keyword>

Summarizes conditions in the OPERation:DEVice event registers.

<keyword>	**Example**
:CONDition?	STAT:OPER:DEV:COND?
:ENABle <bits>	STAT:OPER:DEV:ENAB 16
[:EVENt]?	STAT:OPER:DEV?
:NTRansition <bits>	STAT:OPER:DEV:NTR 16
:PTRansition <bits>	STAT:OPER:DEV:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Unused	
1	2	Unused	
2	4	Unused	
3	8	Unused	
4	16	Sweep Completed	When sweep is complete
5	32	Unused	
6	64	Unused	
7	128	Unused	
8	256	Unused	

9	512	Unused
10	1024	Unused
11	2048	Unused
12	4096	Unused
13	8192	Unused
14	16384	Unused

Status Command Keywords

The following keywords can be appended to the node or nodes that represent the Status register you want to control.

- :CONDition?
- :ENABle
- :ENABle?
- :EVENt?
- :MAP
- :NTRansition
- :PTRansition

Learn about Status Registers

:CONDition?

Monitors the conditions as they occur REAL TIME. That is, a condition may occur, and then clear before the condition is read. Reading this register returns a 16-bit decimal weighted number.

:ENABle <bit>

Enables register bits that will monitored using the **service request (SRQ)** method. (To use the direct read method, you do not have to enable the bit.)

Default value for `STATus:QUEStionable:ENABle` and `STATus:OPERation:ENABle` is 0: No bits enabled.

Default value for all other registers :ENABle <bits> is 32767; ALL BITS ENABLED.

Therefore it is ONLY necessary to send the ENABle keyword if you want to DISABLE some conditions. For example, to enable **ONLY** Trace1 (bit 2) of the LIMIT1 register (disable all other traces) , send: **`STATus:QUEStionable:LIMit1:ENABle 4`**

:ENABle?

Read the enable register to verify the bits that you enabled. Returns a 16 bit weighted sum of the bits that are enabled.

[:EVENt]?

Query only - This is the Default keyword for most registers. Use it to determine if a condition has occured. These bits remain set until they are read or otherwise cleared.

:MAP <bit>,<error>

Associates a bit is the User register with an error number. For example

`STATus:QUEStionable:DEFine:USER2:MAP 0,-113`

0 is the bit that will be set

-113 is the error

When error -113 "Undefined Header" occurs, bit 0 in the USER2 register will be set to 1.

:NTRansition <bits>

Write-Read - Negative Transition register bits set the condition to be set on the Negative going (True to False) transition. Use this register if you are only interested in a condition changing from True to False.

:NTRansition?

queries the register to verify that you set a negative transition.

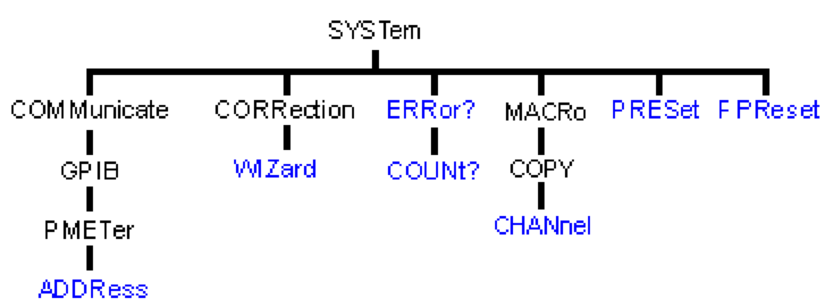
:PTRansition <bits>

Write-Read - Positive Transition register bits set the condition to be set on the False to True transition. Use this register if you are only interested in the change of a condition from False to True.

:PTRansition?

Queries the register to verify that you set a positive transition.

System Commands



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Preset

SYSTem:COMMunicate:GPIB:PMETer:ADDRess <num>

(Read-Write) Specifies the GPIB address of the power meter to be used in a source power calibration.

Parameters

<num>	GPIB address of the power meter. Choose any integer between 0 and 30.
Examples	**SYST:COMM:GPIB:PMET:ADDR 13** system:communicate:gpib:pmeter:address 14
Query Syntax **Return Type**	SYSTem:COMMunicate:GPIB:PMETer:ADDRess? Character
Overlapped? **Default**	No 13

SYSTem:CORRection:WIZard <char>

(Write-only) Launches either the Calibration Wizard or the Version 2 Calibration Kit File Manager dialog box.

Parameters

<char>	Choose from: MAIN - Launches the Calibration Wizard CKIT - Launches the Version 2 Calibration Kit File Manager dialog box. Both display on the PNA screen.
Examples	SYST:CORR:WIZ MAIN system:correction:wizard ckit
Query Syntax	Not Applicable?
Overlapped? **Default**	No Not Applicable

SYSTem:ERRor?

(Read-only) Returns the next error in the error queue. Each time the analyzer detects an error, it places a message in the error queue. When the `SYSTEM:ERROR?` query is sent, one message is moved from the error queue to the output queue so it can be read by the controller. Error messages are delivered to the output queue in the order they were received. The error queue is cleared when any of the following conditions occur:

- When the analyzer is switched ON.
- When the `*CLS` command is sent to the analyzer.
- When all of the errors are read.

If the error queue overflows, the last error is replaced with a `"Queue Overflow"` error. The oldest errors remain in the queue and the most recent error is discarded.

Examples	SYST:ERR? system:error?
Overlapped? **Default**	No Not applicable

SYSTem:ERRor:COUNt?

(Read-only) Returns the number of errors in the error queue.Use SYST:ERR? to read an error.

Examples	SYST:ERR:COUN? system:error:count?
Overlapped?	No

Default	Not applicable

SYSTem:PRESet

(Write-only) Deletes all traces, measurements, and windows. In addition, resets the analyzer to factory defined default settings and creates a S11 measurement named "CH1_S11_1". For a list of default settings, see Default.

If the PNA display is disabled with DISP:ENAB OFF then SYST:PRES will NOT enable the display.

Examples	SYST:PRES system:preset
Overlapped?	No
Default	Not applicable

SYSTem:MACRo:COPY:CHANnel<cnum>[:TO] <num>

(Write-only) Sets up channel <num> as a copy of channel <cnum>.

Parameters

<cnum>	Any existing channel number. If unspecified, value is set to 1.
<num>	Number of the channel which is to become a copy of channel <cnum>.

Examples	`SYST:MACR:COPY:CHAN1 2` system:macro:copy:channel2:to 3
Query Syntax	Not Applicable
Overlapped?	No
Default	Not Applicable

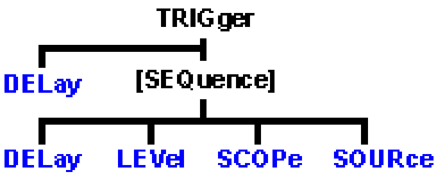
SYSTem:FPReset

(Write-only) Deletes all traces, measurements, and windows. The screen goes blank. This command is used in the factory during instrument programming.

Examples	SYST:FPR system:fpreset
Overlapped?	No
Default	Not applicable

Trigger Commands

Starts or ends a measurement sequence. These commands are an important part of synchronizing measurements.



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Triggering

TRIGger:DELay <num>

(Read-Write) Sets and reads the trigger delay. This delay is only applied while in External Trigger mode. After an external trigger is applied, the start of the sweep is held off to for an amount of time equal to the delay setting plus any inherent latency.

Parameters

<num>	**Float**. Range of delay, from 0 to 1 second.
Examples	`TRIG:DEL .0003` Sets the trigger delay to 300 microseconds. The sweep will not start until approximately 300 microseconds after an external trigger is applied.
Query Syntax	TRIGger:DELay?
Return Type	float num
Overlapped?	No
Default	0

TRIGger[:SEQuence]:DELay <num>

(Read-Write) Sets and reads the trigger delay. This delay is only applied while in External Trigger mode. After an external trigger is applied, the start of the sweep is delayed for the specified delay value plus any inherent latency.

Parameters

<num>	Delay value in seconds. Range from 0 to 1 second.
Examples	TRIG:DEL 1e-3 trigger:sequence:delay .003
Query Syntax	TRIGger[:SEQuence]:DELay?
Return Type	Character
Overlapped?	No
Default	0

TRIGger[:SEQuence]:LEVel <char>

(Read-Write) Triggers either on a **High or Low** level trigger signal. (There is currently no positive or negative edge triggering.) This setting only has an effect when TRIG:SOURce EXTernal is selected.

Parameters

<char>	Choose from: **HIGH -** analyzer triggers on TTL **High** **LOW -** analyzer triggers on TTL **Low**

Examples	TRIG:LEV HIGH trigger:sequence:level low
Query Syntax	TRIGger[:SEQuence]:LEVel?
Return Type	Character
Overlapped?	No
Default	LOW

TRIGger[:SEQuence]:SCOPe <char>

(Read-Write) Specifies whether triggers are applied to all channels or the current channel.

Parameters

<char>	Choose from: **ALL -** triggers all channels. Also sets SENS:SWEep:TRIG:POINt OFF on **ALL** channels. **CURRent -** trigger only one channel at a time. With each trigger signal, the channel is incremented to the next triggerable channel.
Examples	TRIG:SCOP ALL trigger:sequence:scope current
Query Syntax	TRIGger[:SEQuence]:SCOPe?
Return Type	Character
Overlapped?	No
Default	ALL

TRIGger[:SEQuence]:SOURce <char>

(Read-Write) Sets the source of the sweep trigger signal. This command is a super-set of INITiate:CONTinuous, which can NOT set the source to External.

Parameters

<char>	Choose from: **EXTernal -** external (rear panel) source **IMMediate -** internal source sends continuous trigger signals **MANual** - sends one trigger signal when manually triggered from the front panel or INIT:IMM is sent.
Examples	TRIG:SOUR EXT trigger:sequence:source immediate
Query Syntax	TRIGger[:SEQuence]:SOURce?
Return Type	Character
Overlapped?	No
Default	IMMediate

SCPI Examples

SCPI Example Programs

- Catalog Measurements
- Create a Measurement
- Setup Sweep Parameters
- Setup the Display
- Perform a Calibration
- Perform a Guided Cal
- Perform a Source Power Cal
- Perform a Sliding Load Cal
- Perform an ECAL Calibration
- Perform an ECAL Confidence Check
- Getting and Putting Data
- Establish a VISA Session
- Status Reporting
- Modify a Calibration Kit
- GPIB using Visual C++
- Create a Custom Power Meter Driver
- PNA as Controller and Talker/Listener

Catalog Measurements using SCPI

This Visual Basic Program does the following:

- Catalogs the currently defined measurements, windows, and traces
- Selects a measurement for further definition
- Adds a Title to the window

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

```
Dim Meas as String
 Dim Win as String
 Dim Trace as String

 'Read the current measurements in Channel 1
 GPIB.Write "CALCulate1:PARameter:CATalog?"
 Meas = GPIB.Read
 MsgBox ("Ch1 Measurments: " & Meas)

 'Read the current windows
 GPIB.Write "DISPlay:CATalog?"
```

```
 Win = GPIB.Read
 MsgBox ("Windows: " & Win)

 'Read current traces in window 1
 GPIB.Write "DISPlay:WINDow1:CATalog?"
 Trace = GPIB.Read
 MsgBox ("Traces in Window1: " & Win)
```

Create a Measurement using SCPI

This Visual Basic program creates a new S21 measurement and displays it on the display. Use the links to see the command details.

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

```
'Preset the analyzer
 GPIB.Write "SYSTem:PReset"

 ' Turn on window 1 - if new, creates it
 GPIB.Write "DISPlay:WINDow1:STATE ON"

 'Define a measurement name, parameter
 GPIB.Write "CALCulate:PARameter:DEFine 'MyMeas',S21"

 'Associate ("FEED") the measurement name ('MyMeas') to WINDow (1), and
give the new TRACe a number (1).
 GPIB.Write "DISPlay:WINDow1:TRACe1:FEED 'MyMeas'"
```

Create a Custom Power Meter Driver

Note: This topic requires that you have a working knowledge of Visual Basic.

This topic will help you create your own power meter driver for use with Source Power Calibration on the PNA. If you are using one of the following Power Meters to perform a Source Power Calibration, you do NOT need to create your own driver:

E4416A, E4417A, E4418A/B, E4419A/B, 437B, 438A, EPM-441A, EPM-442A

Your Power Meter driver will be created from a template written in Visual Basic using VISA over the GPIB bus.

Note: This procedure applies to Visual Basic 6.0. Applicability to Visual Basic .NET has not yet been investigated.

- Prepare Template Files
- Modify Template Files
- Compile, Copy, and Register, Your New Driver

- Test Your new Driver

Other SCPI Example Programs

Prepare Template Files

1. Copy all the files from the PNA hard drive C:\Program Files\Agilent\Network Analyzer\Automation\Power Meter Driver Template folder, to a folder on your development PC.
2. In Visual Basic click **File**, then **Open Project…**, find **MyPowerMeter.vbp** (a file you copied from the PNA). Click **Open**. This is a VB ActiveX EXE template, which you will fill in to become your driver.
3. Click **Project**, then **MyPowerMeter Properties**. Click the **General** tab.
4. Overwrite the Project Name with a name of your own choosing. This will be the name of your driver's type library (also the default name of your exe).

   Note If the name of your exe does not match the VB Project Name with which it was compiled, registration of the exe on the PNA will not succeed.

5. Set the Project Description. After building your driver if you wish to test it using VB, this is the string that will show up in the VB References list of your test project, and also in the lower pane of the VB Object Browser.
6. Set the Thread Pool size to 1 thread.
7. Click **OK** to close the project properties dialog.
8. From the VB **Project** menu, click **References…** Ensure that **Agilent PNA Power Meter 1.0 Type Library** and **VISA Library** are checked. Click **OK.**

   Note: Agilent's implementation of VISA is installed as part of the Agilent I/O Libraries on the PNA. For help on VISA, go to the Windows Start button on your PNA, select Programs, Agilent IO Libraries, VISA Help.

Modify Template Files

From Visual Basic **View** menu click **Project Explorer**. Expand the **Modules** and **Class Modules** folders. Ensure there is one module (WinAPI) and one class module (PowerMeter).

Let's look at the WinAPI module first.

1. In the **Project Explorer** window, click **WinAPI**.
2. From the **View** menu click **Code**.

   There is only one line of code you should need to modify in this module: the value of the string constant named sIDSEARCH. The comments preceding the declaration of that string describe how to change it. The rest of this module contains functions which will use the Microsoft Windows API to insure proper registration of your driver on the PNA. If you know of other Windows API functions you feel might be helpful to call from within your PowerMeter class module (to help in formatting data, for example), this module would be the place to declare them.

   Now let's look at the class module.

1. In the Project Explorer window, click **PowerMeter**.
2. From the **View** menu click **Properties Window.** The **Instancing** property must be set to MultiUse. This allows other applications to create objects from this class, such that one instance of your driver EXE can supply more than one such object at a time.
3. From the **View** menu click **Code.**

   Do NOT modify the Interfaces to IPowerMeter subroutines and functions. PNA source power cal expects to find these interfaces as they are currently defined.

The only members that you need to supply code to are those containing **"Your code here"** comments.

In addition, comments have been provided at the beginning of each member to describe the information that member needs to be read from or written to the power meter.

To get an idea of how communicate with the power meter using the VISA functions **viWrite** and **viRead**, examine the code which has been implemented for you in IPowerMeter_Connect, IPowerMeter_QueryMeter, and IPowerMeter_WriteMeter.

Compile, Copy, and Register Your New Driver

When your driver is ready to run, you will first need to compile it into an EXE.

From the File menu select **Make exe.**

After compiling, the following will instruct VB to use the same ID (GUID) every time you re-compile your project.

1. From the **Project** menu, click **PowerMeter Properties**.
2. On the **Component** tab, select **Binary Compatibility** and click **...**
3. Browse to and select your project EXE. Click **Open.**
4. Click **OK** to close **Project Properties.**
5. Save your project.
6. Copy your driver EXE file to a folder on your PNA (do NOT use C:\Program Files\Agilent\Network Analyzer\Automation\Power Meter Driver Template folder).
7. Run the EXE file. A message box will pop up reporting whether or not registration was successful. If not successful, it will make a suggestion on what to fix.

When your driver is properly registered, PNA Source Power Cal should be able to associate it with the ID string of your power meter.

Test Your Power Meter Driver

We have also provided a Visual Basic project to test your new Power Meter driver. This project individually calls every IPowerMeter method and property in your driver to verify that it performs correctly. Before running the test your PC and PNA must be configured to communicate using DCOM.

1. Connect your PC and the PNA to LAN.
2. Add your PC logon to the PNA. Both logons and password must match to communicate using DCOM. See Additional PNA users.
3. Configure your driver using DCOM Config on the PNA. This will give you permission to launch and access the driver. See Configure for COM-DCOM Programming.

Modify the Test Project

1. In Visual Basic click **File**, then **Open Project…**, find **MyPowerMeterTest.vbp** (a file you copied from the PNA). Click **Open**.
2. From the **Project** menu, click **References…** From the list, find and check your new Power Meter Driver. (It should have been registered on your PC when you successfully made your driver EXE.) Click **OK.**
3. From the **View** menu click **Code.**
4. Modify the **CreateObject** line as follows:
   Replace **MyPowerMeter** with the Project Name that you chose for your driver

Replace **MyPNA** with the Computer Name of your PNA.
For example:

```
Set PowerMeterObj = CreateObject("AcmeBrand.PowerMeter", "AGILENT-
PNA123")
```

(This assumes that you kept **PowerMeter** as class module name in your driver.)

Run the Test Project

Ensure your power meter is connected to the PNA with a GPIB cable.

Put the PNA in system controller mode:

1. From the PNA **System** menu point to **Configure** then click **SICL/GPIB**.
2. In the GPIB box click **System Controller**.

Run the test project. If there are no errors, the driver is created successfully. If there are errors, try to figure out what went wrong and fix it. Then re-compile, re-copy the .exe to the PNA, and re-run the test. You should not need to re-register the driver or re-modify the test program.

ECALConfidence Check using SCPI

This Visual Basic program performs a complete ECAL confidence check.

To run this program, you need:

- An established GPIB interface connection
- Agilent's VISA or National Instrument's VISA installed on your PC
- The module visa32.bas added to your VB project.
- A form with two buttons: cmdRun and cmdQuit
- A calibrated S11 1-port or N-port measurement active on Channel 1
- Window 1 is visible

See Other SCPI Example Programs

```
'Session to VISA Default Resource Manager
Private defRM As Long
'Session to PNA
Private viPNA As Long
'VISA function status return code
Private status As Long

Private Sub Form_Load()
   defRM = 0
End Sub

Private Sub cmdRun_Click()
'String to receive data from the PNA
Dim strReply As String * 200

' Open the VISA default resource manager
status = viOpenDefaultRM(defRM)
If (status < VI_SUCCESS) Then HandleVISAError
```

```
' Open a VISA session (viPNA) to the PNA at GPIB address 16.
status = viOpen(defRM, "GPIB0::16::INSTR", 0, 0, viPNA)
If (status < VI_SUCCESS) Then HandleVISAError

' Need to set the VISA timeout value to give all our GPIB Reads
' sufficient time to complete before a timeout error occurs.
' For this example, let's try setting the limit to
' 10000 milliseconds (10 seconds).
status = viSetAttribute(viPNA, VI_ATTR_TMO_VALUE, 10000)
If (status < VI_SUCCESS) Then HandleVISAError

' Get the catalog of all the measurements currently on Channel 1.
status = myGPIBWrite(viPNA, "CALC1:PAR:CAT?")
If (status < VI_SUCCESS) Then HandleVISAError
status = myGPIBRead(viPNA, strReply)
If (status < VI_SUCCESS) Then HandleVISAError

' If an S11 measurement named "MY_S11" doesn't already exist,
' then create it.
If InStr(strReply, "MY_S11") = 0 Then
   status = myGPIBWrite(viPNA, "CALC1:PAR:DEF MY_S11,S11")
   If (status < VI_SUCCESS) Then HandleVISAError
End If
strReply = ""

' Get the catalog of all the trace numbers currently active
' in Window 1.
status = myGPIBWrite(viPNA, "DISP:WIND1:CAT?")
If (status < VI_SUCCESS) Then HandleVISAError
status = myGPIBRead(viPNA, strReply)
If (status < VI_SUCCESS) Then HandleVISAError

' If a trace number 4 already exists in Window 1, then this
' will remove it.
If InStr(strReply, "4") > 0 Then
   status = myGPIBWrite(viPNA, "DISP:WIND1:TRAC4:DEL")
   If (status < VI_SUCCESS) Then HandleVISAError
End If

' Set trace number 4 to MY_S11.
status = myGPIBWrite(viPNA, "DISP:WIND1:TRAC4:FEED MY_S11")
If (status < VI_SUCCESS) Then HandleVISAError

' Set up trace view so we are viewing only the data trace.
status = myGPIBWrite(viPNA, "DISP:WIND1:TRAC4 ON")
If (status < VI_SUCCESS) Then HandleVISAError
status = myGPIBWrite(viPNA, "DISP:WIND1:TRAC4:MEM OFF")
If (status < VI_SUCCESS) Then HandleVISAError

' Select MY_S11 as the measurement to be used for the
' Confidence Check.
status = myGPIBWrite(viPNA, "SENS1:CORR:CCH:PAR MY_S11")
If (status < VI_SUCCESS) Then HandleVISAError

' Acquire the S11 confidence check data from ECal Module A
' into the memory buffer (asking for an OPC reply when it's done).
status = myGPIBWrite(viPNA, "SENS1:CORR:CCH:ACQ ECALA;*OPC?")
If (status < VI_SUCCESS) Then HandleVISAError

' The PNA sends an OPC reply ("+1") when the confidence data
' acquisition into memory is complete, so this Read is waiting on
```

```
' the reply until it is received.
status = myGPIBRead(viPNA, strReply)
If (status < VI_SUCCESS) Then HandleVISAError

' Turn on trace math so the trace shows data divided by memory.
' You can be confident the S11 calibration is reasonably good if
' the displayed trace varies no more than a few tenths of a dB
' from 0 dB across the entire span.
status = myGPIBWrite(viPNA, "CALC1:MATH:FUNC DIV")
If (status < VI_SUCCESS) Then HandleVISAError
End Sub

Private Sub cmdQuit_Click()
' Turn off trace math
status = myGPIBWrite(viPNA, "CALC1:MATH:FUNC NORM")
If (status < VI_SUCCESS) Then HandleVISAError

' Conclude the confidence check to set the ECal module
' back to it's idle state.
status = myGPIBWrite(viPNA, "SENS1:CORR:CCH:DONE")
If (status < VI_SUCCESS) Then HandleVISAError

' Close the resource manager session (which also closes
' the session to the PNA).
If defRM <> 0 Then Call viClose(defRM)

' End the program
End
End Sub

Private Function myGPIBWrite(ByVal viHandle As Long, ByVal strOut As
String) As Long
' The "+ Chr$(10)" appends an ASCII linefeed character to the output,
for
' terminating the write transaction.
myGPIBWrite = viVPrintf(viHandle, strOut + Chr$(10), 0)
End Function

Private Function myGPIBRead(ByVal viHandle As Long, strIn As String) As
Long
myGPIBRead = viVScanf(viHandle, "%t", strIn)
End Function

Sub HandleVISAError()
Dim strVisaErr As String * 200
Call viStatusDesc(defRM, status, strVisaErr)
MsgBox "*** Error : " + strVisaErr, vbExclamation
End
End Sub
```

ECALibrate using SCPI

The following program does an Electronic Calibration using an Agilent ECAL module. These commands do the following:

- Acquire the standards
- Move the error terms back into the analyzer
- Enable the calibration

Note: A separate :SENS:CORR:COLL:SAVE is not needed.

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

```
Private Sub Command5_Click()
 'Turn off continuous sweep
 GPIB.Write "INITiate:CONTinuous OFF"

 'ECal full 1 port and 2 port
 'This program assumes you have already set up the analyzer for an S11
measurement over the frequency range, power, etc. that you want.

 'Select the Ecal "Kit"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT 99"

 'Choose a Calibration Type (comment out one of these)
 GPIB.Write "SENSe:CORRection:COLLect:METHod refl3"
 GPIB.Write "SENSe:CORRection:COLLect:METHod SPARSOLT"

 'Enable or disable (comment out one) measurement of isolation
 GPIB.Write "SENSe:CORRection:ISOLation ON"
 GPIB.Write "SENSe:CORRection:ISOLation OFF"

 'Prompt for the ECal module
 MsgBox ("Connect ECal module to Port 1, then press enter")

 'Acquire and store the calibration terms - return (*OPC) when finished
 GPIB.Write "SENSe:CORRection:COLLect:ACQuire ECALA;*OPC?"
 X = GPIB.Read
 MsgBox ("Done with calibration.")

 End Sub
```

Establish a VISA Session

This Visual Basic program demonstrates how to send a SCPI command using VISA and the Agilent IO libraries. To run this program, you need:

- Your PC and PNA both connected to a LAN (for communicating with each other).
- The SICL and VISA components of Agilent's I/O Libraries software installed on your PC. Both are included when you install the software, unless you already have another vendor's VISA installed. Then specify Full SICL and VISA installation to overwrite the other vendor's VISA.
- The module visa32.bas added to your VB project. After you install VISA, the module will be located at C:\VXIPNP\WINNT (or equivalent)\INCLUDE\Visa32.bas
- A form with two buttons: cmdRun and cmdQuit.
- Your PC configured to be a VISA LAN Client, and the SICL Server capability enabled on the PNA. See Configure for VISA and SICL

See Other SCPI Example Programs

Note: This example is a piece of a larger VISA program that performs a source power calibration.

```
'Session to VISA Default Resource Manager
 Private defRM As Long
 'Session to PNA
 Private viPNA As Long
 'VISA function status return code
 Private status As Long

 Private Sub Form_Load()
 defRM = 0
 End Sub

 Private Sub cmdRun_Click()
 ' String to receive data from the PNA.
 ' Dimensioned large enough to receive scalar comma-delimited values
 ' for 21 frequency points (20 ASCII characters per point)
 Dim strReply As String * 420

 ' Open the VISA default resource manager
 status = viOpenDefaultRM(defRM)
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Open a VISA session (viPNA) to the SICL LAN server
 ' at "address 16" on the PNA pointed to by the "GPIB0"
 ' VISA LAN Client on this PC.
 ' CHANGE GPIB0 TO WHATEVER YOU PNA IS SET TO
 status = viOpen(defRM, "GPIB0::16::INSTR", 0, 0, viPNA)
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Need to set the VISA timeout value to give all our calls to
 ' myGPIBRead sufficient time to complete before a timeout
 ' error occurs.
 ' For this example, let's try setting the limit to
 ' 30000 milliseconds (30 seconds).
 status = viSetAttribute(viPNA, VI_ATTR_TMO_VALUE, 30000)
 If (status < VI_SUCCESS) Then HandleVISAError



 ' Preset the PNA
 status = myGPIBWrite(viPNA, "SYST:PRES")
 If (status < VI_SUCCESS) Then HandleVISAError



 ' Print the data using a message box
 MsgBox strReply
 End Sub

 Private Sub cmdQuit_Click()
 ' Close the resource manager session (which also closes
 ' the session to the PNA).
 If defRM <> 0 Then Call viClose(defRM)

 ' End the program
 End
 End Sub

 Private Function myGPIBWrite(ByVal viHandle As Long, ByVal strOut As
String) As Long
 ' The "+ Chr$(10)" appends an ASCII linefeed character to the
 ' output, for terminating the write transaction.
 myGPIBWrite = viVPrintf(viHandle, strOut + Chr$(10), 0)
```

```
 End Function

 Private Function myGPIBRead(ByVal viHandle As Long, strIn As String) As
Long
 myGPIBRead = viVScanf(viHandle, "%t", strIn)
 End Function

 Sub HandleVISAError()
 Dim strVisaErr As String * 200
 Call viStatusDesc(defRM, status, strVisaErr)
 MsgBox "*** Error : " + strVisaErr, vbExclamation
 End
 End Sub
```

Getting and Putting Data using SCPI

This Visual Basic Program does the following:

- Reads data from the analyzer
- Puts the data back into memory
- To see the data on the analyzer after running the program, from the front panel click: **Trace - Math/Memory - Memory Trace**

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

Note: To change the read and write location of data, removing the comment from the beginning of ONE of the lines, and replace the comment in the beginning of the SDATA and SMEM lines.

```
Private Sub ReadWrite_Click()
 Dim i As Integer
 Dim t As Integer
 Dim q As Integer
 Dim dat As String
 Dim cmd As String
 Dim datum() As Double

 GPIB.Configure
 GPIB.Write "SYSTem:PRESet;*wai"

 'Select the measurement
 GPIB.Write "CALCulate:PARameter:SELect 'CH1_S11_1'"

 'Read the number of data points
 GPIB.Write "SENSe1:SWEep:POIN?"
 numpts = GPIB.Read

 'Turn continuous sweep off
 GPIB.Write "INITiate:CONTinuous OFF"

 'Take a sweep
 GPIB.Write "INITiate:IMMediate;*wai"

 'Ask for the Data

 'PICK ONE OF THESE LOCATIONS TO READ
```

```
'GPIB.Write "CALCulate:DATA? FDATA" 'Formatted Meas
'GPIB.Write "CALCulate:DATA? FMEM" 'Formatted Memory
GPIB.Write "CALCulate:DATA? SDATA" 'Corrected, Complex Meas
'GPIB.Write "CALCulate:DATA? SMEM" 'Corrected, Complex Memory
'GPIB.Write "CALCulate:DATA? SCORR1" 'Error-Term Directivity

'Number of values returned per data point
'q = 1 ' Pick this if reading FDATA or FMEM
q = 2 ' Otherwise pick this

'Parse the data
ReDim datum(q, numpts)
For i = 0 To numpts - 1
  For t = 0 To q - 1
  'Read the Data
  dat = GPIB.Read(20)
  'Parse it into an array
  datum(t, i) = Val(dat)
  Next t
Next i

'PUT THE DATA BACK IN
GPIB.Write "format ascii"

'PICK ONE OF THESE LOCATIONS TO PUT THE DATA
'cmd = "CALCulate:DATA FDATA," 'Formatted Meas
'cmd = "CALCulate:DATA FMEM," 'Formatted Memory
'cmd = "CALCulate:DATA SDATA," 'Corrected, Complex Meas
cmd = "CALCulate:DATA SMEM," 'Corrected, Complex Memory
'cmd = "CALCulate:DATA SCORR1," 'Error-Term Directivity

For i = 0 To numpts - 1
  For t = 0 To q - 1
  If i = numpts - 1 And t = q - 1 Then
  cmd = cmd & Format(datum(t, i))
  Else
  cmd = cmd & Format(datum(t, i)) & ","
  End If
  Next t
Next i

GPIB.Write cmd
End Sub
```

GPIB using Visual C++

See Other SCPI Example Programs

```
/*
  * This example assumes the user's PC has a National Instruments GPIB
board. The example is comprised of three basic parts:
  *
  * 1. Initialization
  * 2. Main Body
  * 3. Cleanup
  *
```

```
  * The Initialization portion consists of getting a handle to the PNA
and then doing a GPIB clear of the PNA.
  *
  * The Main Body consists of the PNA SCPI example.
  *
  * The last step, Cleanup, releases the PNA for front panel control.
  */

 #include <stdio.h>
 #include <stdlib.h>

 /*
  * Include the WINDOWS.H and DECL-32.H files. The standard Windows
  * header file, WINDOWS.H, contains definitions used by DECL-32.H and
  * DECL-32.H contains prototypes for the NI GPIB routines and
constants.
  */
 #include <windows.h>
 #include "decl-32.h"

 #define ERRMSGSIZE 1024 // Maximum size of SCPI command string
 #define ARRAYSIZE 1024 // Size of read buffer

 #define BDINDEX 0 // Board Index of GPIB board
 #define PRIMARY_ADDR_OF_PNA 16 // GPIB address of PNA
 #define NO_SECONDARY_ADDR 0 // PNA has no Secondary address
 #define TIMEOUT T10s // Timeout value = 10 seconds
 #define EOTMODE 1 // Enable the END message
 #define EOSMODE 0 // Disable the EOS mode

 int pna;
 char ValueStr[ARRAYSIZE + 1];
 char ErrorMnemonic[21][5] = {"EDVR", "ECIC", "ENOL", "EADR", "EARG",
   "ESAC", "EABO", "ENEB", "EDMA", "",
   "EOIP", "ECAP", "EFSO", "", "EBUS",
   "ESTB", "ESRQ", "", "", "", "ETAB"};


 void GPIBWrite(char* SCPIcmd);
 char *GPIBRead(void);
 void GPIBCleanup(int Dev, char* ErrorMsg);


 int main()
 {

 char *opc;
 char *result;
 char *value;

 /*
  * =========================================
 * INITIALIZATION SECTION
  * =========================================
  */

   /*
   * The application brings the PNA online using ibdev. A device
handle,pna, is returned and is used in all subsequent calls to the PNA.
   */
```

```
   pna = ibdev(BDINDEX, PRIMARY_ADDR_OF_PNA, NO_SECONDARY_ADDR,
   TIMEOUT, EOTMODE, EOSMODE);
   if (ibsta & ERR)
   {
   printf("Unable to open handle to PNA\nibsta = 0x%x iberr = %d\n",
   ibsta, iberr);
   return 1;
   }

   /*
   * Do a GPIB Clear of the PNA. If the error bit ERR is set in ibsta,
call GPIBCleanup with an error message.
   */
   ibclr (pna);
   if (ibsta & ERR)
   {
   GPIBCleanup(pna, "Unable to perform GPIB clear of the PNA");
   return 1;
   }

 /*
  * ==========================================
  * MAIN BODY SECTION
  * ==========================================
  */

   // Reset the analyzer to instrument preset
   GPIBWrite("SYSTem:FPRESET");

   // Create S11 measurement
   GPIBWrite("CALCulate1:PARameter:DEFine 'My_S11',S11");

   // Turn on Window #1
   GPIBWrite("DISPlay:WINDow1:STATe ON");

   // Put a trace (Trace #1) into Window #1 and 'feed' it from the
measurement
   GPIBWrite("DISPlay:WINDow1:TRACe1:FEED 'My_S11'");

   // Setup the channel for single sweep trigger
   GPIBWrite("INITiate1:CONTinuous OFF;*OPC?");
   opc = GPIBRead();
   GPIBWrite("SENSe1:SWEep:TRIGger:POINt OFF");

   // Set channel parameters
   GPIBWrite("SENSe1:SWEep:POINts 11");
   GPIBWrite("SENSe1:FREQuency:STARt 1000000000");
   GPIBWrite("SENSe1:FREQuency:STOP 2000000000");

   // Send a trigger to initiate a single sweep
   GPIBWrite("INITiate1;*OPC?");
   opc = GPIBRead();

   // Must select the measurement before we can read the data
   GPIBWrite("CALCulate1:PARameter:SELect 'My_S11'");

   // Read the measurement data into the "result" string variable
   GPIBWrite("FORMat ASCII");
   GPIBWrite("CALCulate1:DATA? FDATA");
   result = GPIBRead();
```

```
  // Print the data to the display console window
printf("S11(dB) - Visual C++ SCPI Example for PNA\n\n");
  value = strtok(result, ",");
while (value != NULL)
  {
  printf("%s\n", value);
  value = strtok(NULL, ",");
  }

/*
 * ========================================
 * CLEANUP SECTION
 * ========================================
 */

  /* The PNA is returned to front panel control. */
  ibonl(pna, 0);

  return 0;
}

/*
 * Write to the PNA
 */
void GPIBWrite(char* SCPIcmd)
{
int length;
char ErrorMsg[ERRMSGSIZE + 1];
length = strlen(SCPIcmd) ;

  ibwrt (pna, SCPIcmd, length);
  if (ibsta & ERR)
  {
  strcpy(ErrorMsg, "Unable to write this command to PNA:\n");
  strcat(ErrorMsg, SCPIcmd);

  GPIBCleanup(pna, ErrorMsg);
  exit(1);
  }
}

/*
 * Read from the PNA
 */
char* GPIBRead(void)
{
  ibrd (pna, ValueStr, ARRAYSIZE);
  if (ibsta & ERR)
  {
  GPIBCleanup(pna, "Unable to read from the PNA");
  exit(1);
  }
else
  return ValueStr;
}

/*
 * After each GPIB call, the application checks whether the call
succeeded. If an NI-488.2 call fails, the GPIB driver sets the
corresponding bit in the global status variable. If the call failed,
```

```
this procedure prints an error message, takes the PNA offline and exits.
  */
 void GPIBCleanup(int Dev, char* ErrorMsg)
 {
   printf("Error : %s\nibsta = 0x%x iberr = %d (%s)\n",
   ErrorMsg, ibsta, iberr, ErrorMnemonic[iberr]);
   if (Dev != -1)
   {
   printf("Cleanup: Returning PNA to front panel control\n");
   ibonl (Dev, 0);
   }
 }
```

Modify a Calibration Kit using SCPI

This Visual Basic program:

- Modifies Calibration kit number 3
- Completely defines standard #4 (thru)

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

```
'Modifying cal kit number 3
 Calkitnum = 3

 'Designate the kit selection to be used for performing cal's
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:SELect " & Val(Calkitnum)

 'Reset to factory default values.
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:RESet " & Val(Calkitnum)

 'Name this kit with your own name
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:NAME 'My Cal Kit'"

 'Assign standard numbers to calibration classes
 'Set Port 1, class 1 (S11A) to be standard #8
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer1 8"
 'Set Port 1, class 2 (S11B) to be standard #7
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer2 7"
 'Set Port 1, class 3 (S11C) to be standard #3
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer3 3"
 'Set Port 1, class 4 (S21T) to be standard #4
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer4 4"
 'Set Port 2, class 1 (S22A) to be standard #8
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer5 8"
 'Set Port 2, class 2 (S22B) to be standard #7
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer6 7"
 'Set Port 2, class 3 (S22C) to be standard #3
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer7 3"
 'Set Port 2, class 4 (S12T) to be standard #4
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:ORDer8 4"

 'Set up Standard #4 completely
 'Select Standard #4; the rest of the commands act on it
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard 4"
```

```
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:FMIN 300KHz"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:FMAX 9GHz"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:IMPedance 50"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:DELay 1.234 ns"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:LOSS 23e6"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:C0 0"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:C1 1"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:C2 2"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:C3 3"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:L0 10"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:L1 11"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:L2 12"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:L3 13"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:LABel 'My Special
Thru'"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:TYPE THRU"
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:STANdard:CHARacteristic Coax"
```

Perform a 2-Port Calibration using SCPI

This Visual Basic program does a Full 2-Port Calibration, including Isolation, using ONE set of calibration standards.

To run this program, you need:

- An established GPIB interface connection
- A 2-port measurement set up with desired frequency range, power, and so forth, ready to be calibrated.
- The THRU and Isolation standard definitions apply in both directions

See Other SCPI Example Programs

```
Sub SOLTCal()
 'Turn off continuous sweep
 GPIB.Write "INITiate:CONTinuous OFF"

 'Turn off two sets of standards
 GPIB.Write ":SENSe:CORRection:TSTandards OFF"

 'Turn isolation acquisition on
 GPIB.Write "SENSe:CORRection:ISOLation ON

 'Select 2-Port Calibration
 GPIB.Write "SENSe:CORRection:COLLect:METHod SPARSOLT"

 'Set acquisition to FORWARD
 GPIB.Write "SENSe:CORRection:COLLect:SFORward ON

 'Select a cal kit
 Calkitnum = 3
 GPIB.Write "SENSe:CORRection:COLLect:CKIT:SELect " & Val(Calkitnum)

 'Measure the standards in forward direction
 MsgBox "Connect OPEN to Port 1; then press OK"
 Call Measurestandard("stan1")

 MsgBox "Connect SHORT to Port 1; then press OK"
```

```
Call Measurestandard("stan2")

MsgBox "Connect LOAD to Port 1; then press OK"
Call Measurestandard("stan3")

’Set acquisition to REVERSE
GPIB.Write "SENSe:CORRection:COLLect:SFORward OFF

’Measure the standards in reverse direction
MsgBox "Connect OPEN to Port 2; then press OK"
Call Measurestandard("stan1")

MsgBox "Connect SHORT to Port 2; then press OK"
Call Measurestandard("stan2")

MsgBox "Connect LOAD to Port 2; then press OK"
Call Measurestandard("stan3")

’Turn ON two sets of standards for Thru and Isolation standards
GPIB.Write ":SENSe:CORRection:TSTandards ON"

’Measure the thru and isolation standards
MsgBox "Connect THRU between Ports 1 and 2; then press OK"
Call Measurestandard("stan4")

MsgBox "Disconnect Ports 1 and 2 for isolation; then press OK"
Call Measurestandard("stan5")

’Compute the coefficients and turn on error correction
GPIB.Write "SENSe:CORRection:COLLect:SAVE"

’Resume continuous sweep.
GPIB.Write "INITialize:CONTinuous ON"
End Sub

Sub Measurestandard(Std$)

’Store the results of a sweep as correction data
GPIB.Write "SENSe:CORRection:COLLect " & Std$

’Take a sweep;return when complete
GPIB.Write "INITiate:IMMediate;*OPC?"
OPCreply = GPIB.Read

End Sub
```

Perform a Guided Calibration using SCPI

This Visual Basic program does a Full 2-Port Calibration, including Isolation, using ONE set of calibration standards.

To run this program, you need:

- An established GPIB interface connection
- A 2-port measurement set up with desired frequency range, power, and so forth, ready to

be calibrated.

- The THRU and Isolation standard definitions apply in both directions

See Other SCPI Example Programs

```
Sub GuidedCal()

 Dim prompt As String
 Dim va As String
 Dim dat As String

 GPIB.Configure

 ' Define the connectors
 GPIB.Write "sens:corr:coll:guid:conn:port1 ""Type N (50) male"" "
 GPIB.Write "sens:corr:coll:guid:conn:port2 ""Type N (50) female"" "
 GPIB.Write "sens:corr:coll:guid:conn:port3 ""Not used"" "
 Value = MsgBox("Two Connectors defined.")

 'Define the Cal Kits
 GPIB.Write "sense:corr:coll:guid:ckit:port1 ""85054D"" "
 GPIB.Write "sense:corr:coll:guid:ckit:port2 ""85054D"" "
 Value = MsgBox("Two Kits Defined")

 ' Initiate the calibration and query the number of steps
 GPIB.Write "sens:corr:coll:guid:init"
 GPIB.Write "sens:corr:coll:guid:steps?"
 stp = GPIB.Read(3)
 dat = stp
 Value = MsgBox("Number of steps is " + dat)

 ' Measure the standards
 For i = 1 To stp
 va = i
 step1ofN = "Step " + va + " of " + dat
 GPIB.Write "sens:corr:coll:guid:desc? " + va
 prompt = GPIB.Read(80)
 Value = MsgBox(prompt, vbOKOnly, step1ofN)
 GPIB.Write "sens:corr:coll:guid:acq STAN" + va
 Next i

 ' Save the calibration
 GPIB.Write "sens:corr:coll:guid:save"
 MsgBox ("2-Port cal done!")

 End Sub
```

Perform a Sliding Load Calibration using GPIB

This Visual Basic program does a **only** the sliding load portion of a Calibration.

To run this program, you need:

- An established GPIB interface connection
- A measurement and calibration routine to call this sub-program
- STAN3 set up as a sliding load standard

See Other SCPI Example Programs

```
Sub slide()
 'Measure the sliding load for at least 3 and no more than 7 slides
 'Note that "SLSET" and "SLDONE" must be executed before the actual
acquisition of a slide
 MsgBox "Connect Sliding Load; set to Position 1; then press OK"
 GPIB.Write "SENS:CORR:COLL SLSET"
 GPIB.Write "SENS:CORR:COLL STAN3;"

 MsgBox "Set Sliding Load to position 2; then press OK"
 GPIB.Write "SENS:CORR:COLL SLSET"
 GPIB.Write "SENS:CORR:COLL STAN3;"

 MsgBox "Set Sliding Load to position 3; then press OK"
 GPIB.Write "SENS:CORR:COLL SLDONE"
 GPIB.Write "SENS:CORR:COLL STAN3;"
 End Sub
```

Perform a Source Power Cal using SCPI

Programming the PNA using COM or using SICL/VISA over LAN (as in this example) leaves the PNA free to control GPIB devices as needed. This Visual Basic program demonstrates:

- Performing a source power calibration of Port 2 for Channel 1.
- Reading the calibration data.

To run this program, you need:

- One of the following power meters connected to the PNA through GPIB: E4416A, E4417A, E4418A/B, E4419A/B, 437B, 438A, EPM-441A, EPM-442A

Note: If your power meter is other than these, you can create your own Power Meter Driver using our template.

- Your PC and PNA both connected to a LAN (for communicating with each other).
- The SICL and VISA components of Agilent's I/O Libraries software installed on your PC (both are included when you install the software, unless you already have another vendor's VISA installed. Then specify Full SICL and VISA installation to overwrite the other vendor's VISA.
- The module visa32.bas added to your VB project.
- A form with two buttons: cmdRun and cmdQuit.
- Your PC configured to be a VISA LAN Client, and the SICL Server capability enabled on the PNA.

See Other SCPI Example Programs

Note: Never use GPIB to send the following SCPI command to the PNA: SOURce:POWer:CORRection:COLLect:ACQuire <ASENsor | BSENsor>. Your PC would then be in control of the GPIB, but this command requires the PNA to take GPIB control. The PNA currently does not support "pass control" – a technique whereby GPIB control can be passed back and forth between two machines.

```
'Session to VISA Default Resource Manager
 Private defRM As Long
 'Session to PNA
 Private viPNA As Long
 'VISA function status return code
```

```
Private status As Long

Private Sub Form_Load()
defRM = 0
End Sub

Private Sub cmdRun_Click()
' String to receive data from the PNA.
' Dimensioned large enough to receive scalar comma-delimited values
' for 21 frequency points (20 ASCII characters per point)
Dim strReply As String * 420

' Open the VISA default resource manager
status = viOpenDefaultRM(defRM)
If (status < VI_SUCCESS) Then HandleVISAError

' Open a VISA session (viPNA) to the SICL LAN server
' at "address 16" on the PNA pointed to by the "GPIB0"
' VISA LAN Client on this PC.
status = viOpen(defRM, "GPIB0::16::INSTR", 0, 0, viPNA)
If (status < VI_SUCCESS) Then HandleVISAError

' Need to set the VISA timeout value to give all our calls to
' myGPIBRead sufficient time to complete before a timeout
' error occurs.
' For this example, let's try setting the limit to
' 30000 milliseconds (30 seconds).
status = viSetAttribute(viPNA, VI_ATTR_TMO_VALUE, 30000)
If (status < VI_SUCCESS) Then HandleVISAError

' Set the number of sweep points to 21 on Channel 1.
status = myGPIBWrite(viPNA, "SENS1:SWE:POIN 21")
If (status < VI_SUCCESS) Then HandleVISAError

' Specify the GPIB address of the power meter
' that will be used in performing the calibration.
status = myGPIBWrite(viPNA, "SYST:COMM:GPIB:PMET:ADDR 13")
If (status < VI_SUCCESS) Then HandleVISAError

' Turn use of the loss table OFF (this assumes there is
' virtually no loss in the RF path to the power sensor
' due to a splitter, coupler or adapter).
status = myGPIBWrite(viPNA, "SOUR:POW:CORR:COLL:TABL:LOSS OFF")
If (status < VI_SUCCESS) Then HandleVISAError

' Turn frequency checking OFF (so one power sensor is used for the
entire cal
' acquisition sweep regardless of frequency span).
status = myGPIBWrite(viPNA, "SOUR:POW:CORR:COLL:FCH OFF")
If (status < VI_SUCCESS) Then HandleVISAError

' Specify the cal power level in dBm expected at the desired reference
plane.
status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:LEV -10 DBM")
If (status < VI_SUCCESS) Then HandleVISAError

' Specify the number of power readings per frequency point (i.e.,
averaging factor)
' to be used during the source power cal acquisition.
status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:COLL:AVER:COUN 2")
```

```
 If (status < VI_SUCCESS) Then HandleVISAError

 ’ Specify the method (type of device) that will be used to perform the
cal.
 status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:COLL:METH PMET")
 If (status < VI_SUCCESS) Then HandleVISAError
 ’ Perform the source power cal acquisition sweep using the sensor
attached to
 ' Channel A of the power meter (asking for an OPC reply when it’s
done).  This
 ' assumes that the power sensor is already connected to Port 2 of the
PNA.
 status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:COLL:ACQ ASEN;*OPC?")
 If (status < VI_SUCCESS) Then HandleVISAError

 ' The PNA sends an OPC reply ("+1") when the cal acquisition is
complete, so
 ' this Read is waiting on the reply until it is received.
 status = myGPIBRead(viPNA, strReply)
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Conclude the calibration.  This applies the cal data to PNA channel
memory,
 ' and turns the correction ON for Port 2 on Channel 1,
 ' but does NOT save the calibration.
 status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:COLL:SAVE")
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Prepare for doing data transfer in ASCII format.
 status = myGPIBWrite(viPNA, "FORM:DATA ASCII")
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Read the source power correction data into the strReply string.
 status = myGPIBWrite(viPNA, "SOUR1:POW2:CORR:DATA?")
 If (status < VI_SUCCESS) Then HandleVISAError
 status = myGPIBRead(viPNA, strReply)
 If (status < VI_SUCCESS) Then HandleVISAError

 ' Print the data using a message box
 MsgBox strReply
 End Sub

 Private Sub cmdQuit_Click()
 ' Close the resource manager session (which also closes
 ' the session to the PNA).
 If defRM <> 0 Then Call viClose(defRM)

 ' End the program
 End
 End Sub

 Private Function myGPIBWrite(ByVal viHandle As Long, ByVal strOut As
String) As Long
 ' The "+ Chr$(10)" appends an ASCII linefeed character to the
 ' output, for terminating the write transaction.
 myGPIBWrite = viPrintf(viHandle, strOut + Chr$(10), 0)
 End Function

 Private Function myGPIBRead(ByVal viHandle As Long, strIn As String) As
Long
 myGPIBRead = viVScanf(viHandle, "%t", strIn)
```

```
End Function

Sub HandleVISAError()
Dim strVisaErr As String * 200
Call viStatusDesc(defRM, status, strVisaErr)
MsgBox "*** Error : " + strVisaErr, vbExclamation
End
End Sub
```

PNA as Controller and Talker / Listener

This Visual Basic Program uses VISA to do the following:

- This Visual Basic Program uses VISA to do the following:
- Control the PNA using a VISA LAN Client interface on the PNA.
- Control another instrument using the PNA as GPIB controller.
- Queries both the analyzer and other instrument to identify themselves with *IDN?

Note: This program can be modified to work from a remote PC to control both instruments. In that case, set up the PNA to be a talker/listener.

To run this program, you need to do the following:

- Add module **visa32.bas** to the VB project. It is located on the analyzer at C:\Program Files\HP\VXIPNP\WINNT\Include\VISA32.bas
- Configure the PNA for VISA / SICL
- Set up the PNA to be GPIB system controller.
    1. On the **System** menu, point to **Configure**. Click **SICL / GPIB**
    2. Click **System Controller**
- Connect another instrument to the analyzer through a GPIB cable with Primary address of 13 on GPIB0 interface

See Other SCPI Example Programs

```
Sub main()

'This application run from onboard the PNA
'can control both the PNA and another GPIB instrument.
'
'To run this program the module visa32.bas must be added
'to the project.

'VISA function status return code
Dim status As Long
'Session to Default Resource Manager
Dim defRM As Long
'Session to instrument
Dim viPNA As Long
'Session to other GPIB instrument
Dim viInstrument As Long
'String to hold results
Dim strRes As String * 200
On Error GoTo ErrorHandler

status = viOpenDefaultRM(defRM)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler
```

```
'Open the session to the PNA
status = viOpen(defRM, "GPIB1::16::INSTR", 0, 0, viPNA)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler

'Ask for the PNA's ID.
status = viVPrintf(viPNA, "*IDN?" + Chr$(10), 0)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler

'Read the ID as a string.
status = viVScanf(viPNA, "%t", strRes)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler
'Display the results
MsgBox "PNA is: " + strRes


'Open the session to the other instrument
status = viOpen(defRM, "GPIB0::13::INSTR", 0, 0, viInstrument)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler

'Ask for the instrument's ID.
status = viVPrintf(viInstrument, "*IDN?" + Chr$(10), 0)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler

'Read the ID as a string.
status = viVScanf(viPNA, "%t", strRes)
If (status < VI_SUCCESS) Then GoTo VisaErrorHandler

'Display the results
MsgBox "Other instrument is: " + strRes
' Close the resource manager session (which closes everything)
Call viClose(defRM)
End

ErrorHandler:
'Display the error message
MsgBox "*** Error : " + Error$, MB_ICONEXCLAMATION
End

VisaErrorHandler:
Dim strVisaErr As String * 200
Call viStatusDesc(defRM, status, strVisaErr)
MsgBox "*** Error : " + strVisaErr

End
End Sub
```

Setup Sweep Parameters using SCPI

This Visual Basic program sets up sweep parameters on the Channel 1 measurement.

To run this program, you need:

- An established GPIB interface connection

```
GPIB.Write "SYSTem:PRESet"
 'Select the measurement
 GPIB.Write "CALCulate:PARameter:SELect 'CH1_S11_1'"
 'Set sweep type to linear
 GPIB.Write "SENSe1:SWEep:TYPE LIN"

 'Set IF Bandwidth to 700 Hz
 GPIB.Write "SENSe1:BANDwidth 700"

 'Set Center and Span Freq's to 4 GHz
 GPIB.Write "SENSe1:FREQuency:CENTer 4ghz"
 GPIB.Write "SENSe1:FREQuency:SPAN 4ghz"

 'Set number of points to 801
 GPIB.Write "SENSe1:SWEep:POINts 801"

 'Set sweep generation mode to Analog
 GPIB.Write "SENSe1:SWEep:GENeration ANAL"

 'Set sweep time to Automatic
 GPIB.Write "SENSe1:SWEep:TIME:AUTO ON"

 'Query the sweep time
 GPIB.Write "SENSe1:SWEep:TIME?
 SweepTime = GPIB.Read
```

Setup the Display using SCPI

This Visual Basic program:

- Sets data formatting
- Turns ON the Trace, Title, and Frequency Annotation
- Autoscales the Trace
- Queries Per Division, Reference Level, and Reference Position
- Turn ON and set averaging
- Turn ON and set smoothing

To run this program, you need:

- An established GPIB interface connection

See Other SCPI Example Programs

```
GPIB.Write "SYSTem:PRESet"

 'Select the measurement
 GPIB.Write "CALCulate:PARameter:SELect 'CH1_S11_1'"

 'Set the Data Format to Log Mag
 GPIB.Write ":CALCulate1:FORMat MLOG"
```

```
'Turn ON the Trace, Title, and Frequency Annotation
GPIB.Write "Display:WINDow1:TRACe1:STATe ON"
GPIB.Write "DISPlay:WINDow1:TITLe:STATe ON"
GPIB.Write "DISPlay:ANNotation:FREQuency ON"

'Autoscale the Trace
GPIB.Write "Display:WINDow1:TRACe1:Y:Scale:AUTO"

'Query back the Per Division, Reference Level, and Reference Position
GPIB.Write "DISPlay:WINDow1:TRACe1:Y:SCALe:PDIVision?"
Pdiv = GPIB.Read
GPIB.Write "DISPlay:WINDow1:TRACe1:Y:SCALe:RLEVel?"
Rlev = GPIB.Read
GPIB.Write "DISPlay:WINDow1:TRACe1:Y:SCALe:RPOSition?"
Ppos = GPIB.Read

'Turn ON, and average five sweeps
GPIB.Write "SENSe1:AVERage:STATe ON"
GPIB.Write "SENSe1:AVERage:Count 5"

'Turn ON, and set 20% smoothing aperture
GPIB.Write "CALCulate1:SMOothing:STATe ON"
GPIB.Write "CALCulate1:SMOothing:APERture 20"
```

See Other SCPI Example Programs

Status Reporting using SCPI

This Visual Basic program demonstrates two methods of reading the analyzer's status registers:

- Polled Bit Method - reads the Limit1 register continuously.
- SRQ Method - enables an interrupt of the program when bit 6 of the status byte is set to 1. The program then queries registers to determine if the limit line failed.

To run this program, you need:

- An established GPIB interface connection
- A form with two buttons: Poll and SRQ Method
- A means of causing the limit line to fail, assuming it passes initially.

```
Private Sub Poll_Click()
 ' POLL THE BIT METHOD
 ' Clear status registers
 GPIB.Write "*CLS"

 'Loop FOREVER
 Do
```

```
  DoEvents
  GPIB.Write "STATus:QUEStionable:LIMit1:EVENt?"
  onn = GPIB.Read
Loop Until onn = 2

MsgBox "Limit 1 Failed "
End Sub



Private Sub SRQMethod_Click()
'SRQ METHOD
GPIB.Write "SYSTem:PRESet"
GPIB.Write "CALCulate:PARameter:SELect 'CH1_S11_1'"
'slow down the trace
GPIB.Write "SENS:BWID 150"

'Setup limit line
GPIB.Write "CALC:LIM:DATA 2,3e9,6e9,-2,-2"
GPIB.Write "CALC:LIMit:DISP ON"
GPIB.Write "CALC:LIMit:STATe ON"

' Clear status registers.
GPIB.Write "*CLS;*wai"
' Clear the Service Request Enable register.
GPIB.Write "*SRE 0"
' Clear the Standard Event Status Enable register.
GPIB.Write "*ESE 0"

' Enable questionable register, bit(10) to report to the status byte.
GPIB.Write "STATus:QUEStionable:ENABle 1024"

' Enable the status byte register bit3 (weight 8) to notify controller
GPIB.Write "*SRE 8"

' Enable the onGPIBNotify event
GPIB.NotifyMask = cwGPIBRQS
GPIB.Notify
End Sub

----------------------------------------------------
Private Sub GPIB_OnGPIBNotify(ByVal mask As Integer)
' check to see what failed
' was it the analyzer?
GPIB.Write "*STB?"
onn = GPIB.Read
If onn <> 0 Then
' If yes, then was it the questionable register?
  GPIB.Write "STATus:QUEStionable:EVENt?"
  onn = GPIB.Read
  ' Determine if the limit1 register, bit 8 is set.
  If onn = 1024 Then
  'if yes, then was it trace 1?
  GPIB.Write "STAT:QUES:LIMIT1:EVEN?"
  onn = GPIB.Read
  If onn = 2 Then MsgBox ("Limit Line1 Failed")
  End If
End If
End Sub
```

Learning about SCPI

Learning about GPIB

The following topics can help you learn more about controlling the PNA using SCPI and the GPIB.

- GP-IB Fundamentals
- The Rules and Syntax of SCPI Commands
- Getting Data from the PNA using SCPI
- Configure for VISA and SICL
- Reading the PNA Status Registers
- Understanding SCPI Command Synchronization

GPIB Fundamentals

The General Purpose Interface Bus (GPIB) is a system of hardware and software that allows you to control test equipment to make measurements quickly, accurately, and repeatably. This topic contains the following information:

- The GPIB Hardware Components
- The GPIB / SCPI Programming Elements
- How to Configure for GPIB / SICL
- LCL- RMT Operation Label
- Specifications
- GPIB Interface Capability Codes

Note: All of the topics related to programming assume that you already know how to program, preferably using a language that can control instruments.

Other Topics about GPIB Concepts

The GPIB Hardware Components

The system bus and its associated interface operations are defined by the IEEE 488 standard. The following sections list and describe the main pieces of hardware in a GPIB system:

Instruments

The analyzer is configured as a Talker / Listener by default.

- **Talkers** are instruments that can be addressed to send data to the controller.
- **Listeners** are instruments that can be addressed to receive a command, and then respond to the command. All devices on the bus are required to listen.

GPIB Addresses

Every GPIB instrument must have its own unique address on the bus. The analyzer address (716) consists of two parts:

1. **The Interface select code** (typically 7) indicates which GPIB port in the system controller is used to communicate with the device.

2. **The primary address** (16) is set at the factory. You can change the primary address of any device on the bus to any number between 0 and 30. To change the analyzer address click **System \ Configure \ SICL-GPIB**

The secondary address is sometimes used to allow access to individual modules in a modular instrument system, such as a VXI mainframe. The analyzer does not have secondary addresses.

Controllers

Controllers specify the instruments that will be the talker and listener in a data exchange. The controller of the bus must have a GPIB interface card to communicate on the GPIB.

- The **Active Controller** is the computer or instrument that is currently controlling data exchanges.
- The **System Controller** is the only computer or instrument that can take control and give up control of the GPIB to another computer or instrument, which is then called the active controller.

The PNA can NOT be passed control of the GPIB. However, you can use VISA or SICL over LAN to accomplish this. See this example. You can also accomplish this using COM programming.

Cables

GPIB Cables are the physical link connecting all of the devices on the bus. There are eight data lines in a GPIB cable that send data from one device to another. There are also eight control lines that manage traffic on the data lines and control other interface operations.

You can connect instruments to the controller in any arrangement with the following limitations:

- Do not connect more than 15 devices on any GPIB system. This number can be extended with the use of a bus extension.
- Do not exceed a total of 20 meters of total cable length or 2 meters per device, whichever is less.
- Avoid stacking more than three connectors on the back panel of an instrument. This can cause unnecessary strain on the rear-panel connector.

The GPIB / SCPI Programming Elements

The following software programming elements combine to become a GPIB program:

- GPIB / SCPI Commands
- Programming Statements
- Instrument Drivers

GPIB Commands

The GPIB command is the basic unit of communication in a GPIB system. The analyzer responds to three types of GPIB commands:

1. IEEE 488.1 Bus-management Commands

These commands are used primarily to tell some or all of the devices on the bus to perform certain interface operations.

All of the functions that can be accomplished with these commands can also be done with IEEE 488.2 or SCPI commands. Therefore, these commands are not documented in this Help system. For a complete list of IEEE 488.1 commands refer to the IEEE 488 standard. **Examples** of IEEE 488.1 Commands

- `CLEAR` **– Clears the bus of any pending operations**
- `LOCAL` **– Returns instruments to local operation**

2. IEEE 488.2 Common Commands

These commands are sent to instruments to perform interface operations. An IEEE 488.2 common command consists of a single mnemonic and is preceded by an asterisk (*). Some of the commands have a query form which adds a "?" after the command. These commands ask the instrument for the current setting. See a complete list of the Common Commands that are recognized by the analyzer. **Examples** of IEEE 488.2 Common Commands

- `*OPC  - Operation Complete`
- `*RST  - Reset`
- `*OPT? - Queries the option configuration`

3. SCPI Commands

The Standard Commands for Programmable Instruments (SCPI) is a set of commands developed in 1990. The standardization provided in SCPI commands helps ensure that programs written for a particular SCPI instrument are easily adapted to work with a similar SCPI instrument. SCPI commands tell instruments to do device specific functions. For example, SCPI commands could tell an instrument to make a measurement and output data to a controller. **Examples** of SCPI Commands:

```
CALCULATE:AVERAGE:STATE ON
SENSE:FREQUENCY:START?
```

For more information on SCPI:T

- The Rules and Syntax of SCPI Commands provides more detail of the SCPI command structure.
- SCPI Command Tree is a complete list of the SCPI commands for the analyzer

Programming Statements

SCPI commands are included with the language specific I/O statements to form program statements. The programming language determines the syntax of the programming statements. SCPI programs can be written in a variety of programming languages such as VEE, HP BASIC, or C++. **Example** of a Visual Basic statement:

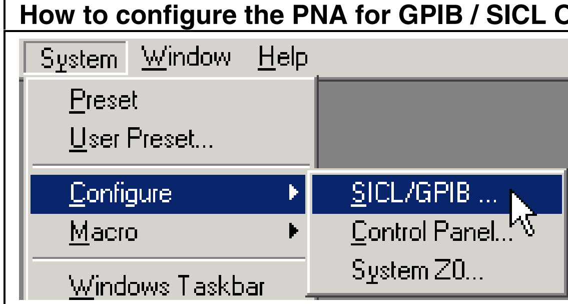
- `GPIB.Write "SOURCE:FREQUENCY:FIXED 1000 MHz"`

Note about examples

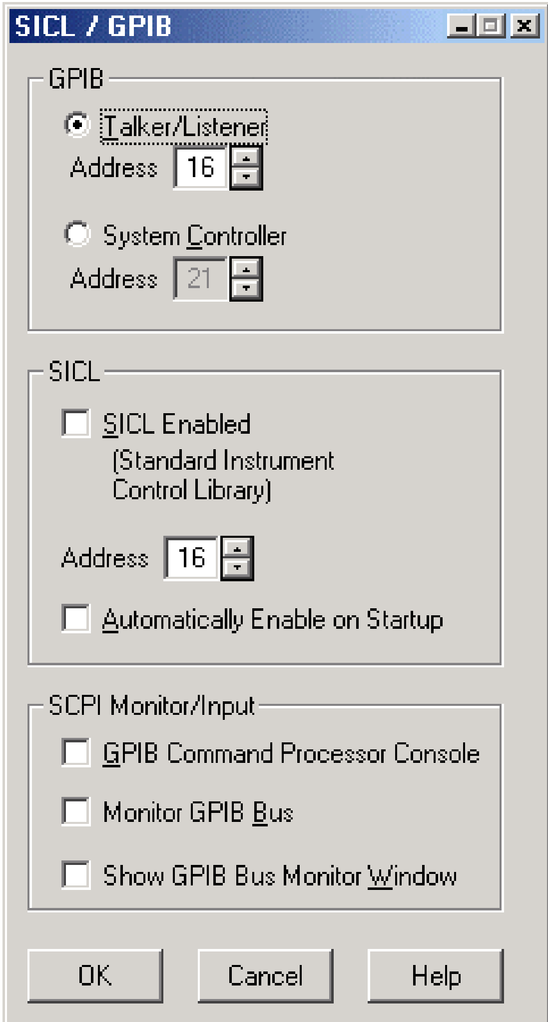
Instrument Drivers

Instrument drivers are subroutines that provide routine functionality and can be reused from program to program. GPIB industry leaders have written standards for use by programmers who develop drivers. When programmers write drivers that comply with the standards, the drivers can be used with predictable results. To comply with the standard, each instrument driver must include documentation describing its functionality and how it should be implemented.

How to configure the PNA for GPIB / SICL Operation



Learn more about using the front panel interface



SICL / GPIB dialog box help

GPIB

Talker/Listener Sets the PNA to receive and send GPIB/SCPI messages to the system controller (external computer).

Talker/Listener Address Sets the PNA talker/listener GPIB address.

System Controller Sets the PNA as the system controller, controlling GPIB communications of external devices. Learn about the PNA as controller.

System Controller Address Sets the PNA system controller GPIB address.

SICL

SICL Enabled When checked, the analyzer is capable of running GPIB programs on its computer to control analyzer functions. The programs must be run from a GPIB-capable programming environment (VEE, Visual Basic). This mode does not allow control of external GPIB instruments. To uncheck this box, exit the PNA application - (Click File, then Exit). The PNA restarts with the SICL enabled box unchecked unless **Automatically Enable on Startup** is checked.

Learn more about Configuring for VISA and SICL.

Address Sets the PNA address.

Automatically Enable on Startup When checked, SICL Enabled is automatically selected when starting the PNA application.

SCPI Monitor / Input

GPIB Command Processor Console Launches a window that is used to send single SCPI/GPIB commands.

Note: Press **Control**+**Z** , then enter, to close the console window.

- Type a valid command, with appropriate arguments and press enter.
- Use the arrow keys to recall previous commands.
- The console window may launch behind the PNA application. Press **Control**+**Tab** to bring the console window to the top.

Monitor GPIB Bus Enables monitoring activity on the GPIB.

Show GPIB Bus Monitor Window Shows and hides the window monitoring GPIB activity.

LCL and RMT Operation

The analyzer **LCL** and **RMT** (Local and Remote) operation labels appear in the lower right corner of the status bar.

Note: The status bar is NOT visible when the analyzer is preset. To make the bar visible, click **View** then **Status Bar**

- **LCL** appears when not under SCPI control
- **RMT** appears when under SCPI control. The RMT label does NOT appear when under COM control

RMT disables the front panel keys except for the **Macro/Local** key. 

Pressing the Macro / Local key returns the analyzer to Local (front panel) operation.

The IEEE488.1 "GTL" (go to local) command also returns the analyzer to Local (front panel) operation.

The IEEE488.1 "LLO" (local lockout) command disables the front panel Local button.

GPIB Specifications

Interconnected devices - Up to 15 devices (maximum) on one contiguous bus.

Interconnection path - Star or linear (or mixed) bus network, up to 20 meters total transmission path length or 2 meters per device, whichever is less.

Message transfer scheme - Byte-serial, bit-parallel, asynchronous data transfer using an interlocking 3-wire handshake.

Maximum data rate - 1 megabyte per second over limited distances, 250 to 500 kilobytes per second typical maximum over a full transmission path. The devices on the bus determine the actual data rate.

Address capability - Primary addresses, 31 Talk and 31 Listen; secondary addresses, 961 Talk and 961 Listen. There can be a maximum of 1 Talker and up to 14 Listeners at a time on a single bus. See also previous section on GPIB addresses.

GPIB Interface Capability Codes

The IEEE 488.1 standard requires that all GPIB compatible instruments display their interface capabilities on the rear panel using codes. The codes on the analyzer, and their related descriptions, are listed below:

SH1	full source handshake capability
AH1	full acceptor handshake capability
T6	basic talker, serial poll, no talk only, unaddress if MLA (My Listen Address)
TEO	no extended talker capability
L4	basic listener, no listen only, unaddress if MTA (My Talk Address)
LEO	no extended listener capability
SR1	full service request capability
RL1	full remote / local capability
PPO	**no parallel poll capability**
DC1	full device clear capability
DT1	full device trigger capability
C1	system controller capability
C2	send IFC (Interface Clear) and take charge controller capability
C3	send REN (Remote Enable) controller capability
C4	respond to SRQ (Service Request)

The Rules and Syntax of SCPI

Most of the commands used for controlling instruments on the GPIB are SCPI commands. The following sections will help you learn to use SCPI commands in your programs.

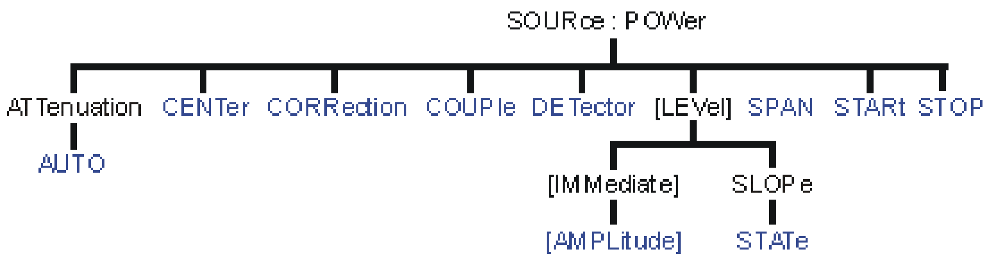
- Branches on the Command Tree
- Command and Query
- Multiple Commands
- Command Abbreviation
- Bracketed (Optional) Keywords
- Vertical Bars (Pipes)
- MIN and MAX Parameters

Other Topics about GPIB Concepts

Branches on the Command Tree

All major functions on the analyzer are assigned keywords which are called ROOT commands. (See GPIB Command Finder for a list of SCPI root commands). Under these root commands are branches that contain one or more keywords. The branching continues until each analyzer function is assigned to a branch. A root command and the branches below it is sometimes known as a subsystem.

For example, the following graphic shows the SOURce subsystem. Under the `SOURce` and `POWer` keywords are several branch commands.



Sometimes the same keyword, such as `STATE`, is used in several branches of the command tree. To keep track of the current branch, the analyzer's command parser uses the following rules:

- **Power On and Reset** - After power is cycled or after `*RST`, the current path is set to the root level commands.
- **Message Terminators** - A message terminator, such as a <NL> character, sets the current path to the root command level. Many programming language output statements send message terminators automatically. Message terminators are described in Sending Messages to the Analyzer.
- **Colon (:)** - When a colon is between two command keywords, it moves the current path down one level in the command tree. For example, the colon in `:SOURCE:POWER` specifies that `POWER` is one level below `SOURCE`. When the colon is the first character of a command, it specifies that the following keyword is a root level command. For example, the colon in `:SOURCE` specifies that source is a root level command.

Note: You can omit the leading colon if the command is the first of a new program line. For example, the following two commands are equivalent:

```
SOUR:POW:ATT:AUTO
:SOUR:POW:ATT:AUTO
```

- **<WSP>** - Whitespace characters, such as <tab> and <space>, are generally ignored. There are two important exceptions:
  - Whitespace inside a keyword, such as `:CALC ULATE`, is not allowed.
  - Most commands end with a parameter. You must use whitespace to separate these ending parameters from commands. **Always refer to the command documentation**. In the following example, there is whitespace between `STATE` and `ON.`

  `CALCULATE1:SMOOTHING:STATE ON`
- **Comma (,)** - If a command requires more than one parameter, you must separate adjacent parameters using a comma. For example, the `SYSTEM:TIME` command requires three values to set the analyzer clock: one for hours, one for minutes, and one for seconds. A message to set the clock to 8:45 AM would be `SYSTEM:TIME 8,45,0.` Commas do not affect the current path.
- **Semicolon(;)** - A semicolon separates two commands in the same message without changing the current path. See Multiple Commands later in this topic.
- **IEEE 488.2 Common Commands** - Common commands, such as `*RST`, are not part of any subsystem. An instrument interprets them in the same way, regardless of the current path setting.

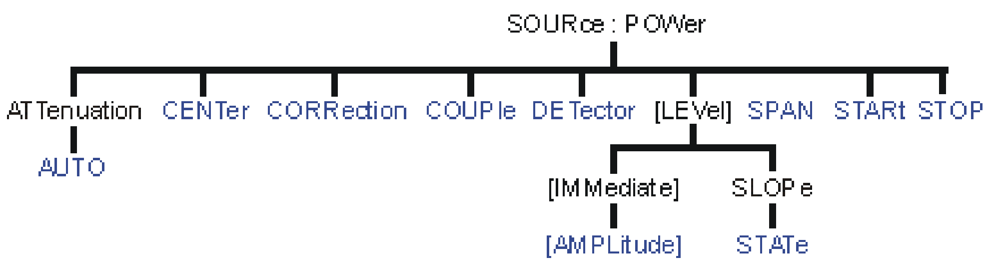
Command and Query

A SCPI command can be an Event command, Query command (a command that asks the analyzer for information), or both. The following are descriptions and examples of each form of command. GPIB Command Finder lists every SCPI command that is recognized by the analyzer, and its form.

Form	Examples
Event commands - cause an action to occur inside the analyzer.	:INITIATE:IMMEDIATE
Query commands - query only; there is no associated analyzer state to set.	:SYSTem:ERRor?
Command and query - set or query an analyzer setting. The query form appends a question mark (?) to the set form	:FORMat:DATA ! Command :FORMat:DATA? ! Query

Multiple Commands

You can send multiple commands within a single program message. By separating the commands with semicolons the current path does not change. The following examples show three methods to send two commands:



1. **Two program messages:**

```
SOURCE:POWER:START 0DBM
 SOURCE:POWER:STOP 10DBM
```

2. **One long message**. A colon follows the semicolon that separates the two commands causing the command parser to reset to the root of the command tree. As a result, the next command is only valid if it includes the entire keyword path from the root of the tree:

```
SOURCE:POWER:START 0DBM;:SOURCE:POWER:STOP 10DBM
```

3. **One short message**. The command parser keeps track of the position in the command tree. Therefore, you can simplify your program messages by including only the keyword at the same level in the command tree.

```
SOURCE:POWER:START 0DBM;STOP 10DBM
```

Common Commands and SCPI Commands

You can send Common commands and SCPI commands together in the same message. (For more information on these types of commands see GP-IB Fundamentals.) As in sending multiple SCPI commands, you must separate them with a semicolon.

Example of Common command and SCPI commands together

```
*RST;SENSE:FREQUENCY:CENTER 5MHZ;SPAN 100KHZ
```

Command Abbreviation

Each command has a long form and an abbreviated short form. The syntax used in this Help system use uppercase characters to identify the short form of a particular keyword. The remainder of the keyword is lower case to complete the long form.

```
SOUR - Short form
 SOURce - Long form
```

Either the complete short form or complete long form must be used for each keyword. However, the keywords used to make a complete SCPI command can be a combination of short form and long form.

The following is **unacceptable** - The first three keywords use neither short or long form.

SOURc:Powe:Atten:Auto on

The following is **acceptable** - All keywords are either short form or long form.

SOUR:POWer:ATT:AUTO on

In addition, the analyzer accepts lowercase and uppercase characters as equivalent as shown in the following equivalent commands:

```
source:POW:att:auto ON
 Source:Pow:Att:Auto on
```

Optional [Bracketed] Keywords

You can omit some keywords without changing the effect of the command. These optional, or default, keywords are used in many subsystems and are identified by brackets in syntax diagrams.

Example of Optional Keywords

The `HCOPy` subsystem contains the optional keyword `IMMediate` at its first branching point. Both of the following commands are equivalent:

```
"HCOPY:IMMEDIATE"
"HCOPY"
The syntax in this Help system looks like this:
HCOPy[:IMMediate]
```

Vertical Bars | Pipes

Vertical bars, or "pipes", can be read as **"or"**. They are used in syntax diagrams to separate alternative parameter options.

Example of Vertical Bars:

SOURce:POWer:ATTenuation:AUTO <on|off>

Either `ON` or `OFF` is a valid parameter option.

MIN and MAX Parameters

The special form parameters "MINimum" and "MAXimum" can be used with **some** commands in the analyzer, as noted in the command documentation. The short form (min) and long form (minimum) of these two keywords are equivalent.

- **MAX**imum refers to the largest value that the function can currently be set to
- **MIN**imum refers to the smallest value that the function can currently be set to.

For example, the following command sets the start frequency to the smallest value that is currently possible:

SENS:FREQ:START MIN

In addition, the max and min values can also be queried for these commands.

For example, the following command returns the smallest value that Start Frequency can currently be set to:

SENS:FREQ:START? MIN

An error will be returned if a numeric parameter is sent that exceeds the MAX and MIN values.

For example, the following command will return an "Out of range" error message.

```
SENS:FREQ:START 1khz
```

Getting Data from the Analyzer

Data is sent from the analyzer in response to program queries. Data can be short response messages, such as analyzer settings, or large blocks of measurement data. This topic discusses how to read query responses and measurement data from the analyzer in the most efficient manner.

- Response Message Syntax
- Clearing the Output Queue
- Response Data Types
- Transferring Measurement Data

Note: Some PCs use a modification of the IEEE floating point formats with the byte order reversed. To reverse the byte order for data transfer into a PC, the FORMat:BORDer command should be used. See GPIB Command Finder for details.

Other Topics about GPIB Concepts

Response Message Syntax

Responses sent from the analyzer contain data, appropriate punctuation, and message terminators.

<NL><^END> is always sent as a response message terminator. Most programming languages handle these terminators transparent to the programmer.

Response messages use commas and semicolons as separators in the following situations:

- a comma separates response data items when a single query command returns multiple values

```
FORM:DATA? 'Query
 ASC, +0 'Analyzer Response
```

- a semicolon separates response data when multiple queries are sent within the same messages

```
SENS:FREQ:STAR?;STOP? --Example Query

+1.23000000E+008; +7.89000000E+008<NL><^END> 'Analyzer Response
```

Clearing the Output Queue

After receiving a query, the analyzer places the response message in it's output queue. Your program should read the response immediately after the query is sent. This ensures that the response is not cleared before it is read. The response is cleared when one of the following conditions occur:

- When the query is not properly terminated with an ASCII carriage return character or the GPIB <^END> message.
- When a second program query is sent.

- When a program message is sent that exceeds the length of the input queue
- When a response message generates more response data than fits in the output queue.
- When the analyzer is switched ON.

Response Data Types

The analyzer sends different response data types depending on the parameter being queried. You need to know the type of data that will be returned so that you can declare the appropriate type of variable to accept the data. For more information on declaring variables see your programming language manual. The GPIB Command Finder lists every GPIB command and the return format of data in response to a query. The analyzer returns the following types of data:

- Numeric Data
- Character Data
- String Data
- Block Data

Numeric Data

The analyzer sends ASCII character data that looks like numeric data. All numeric data sent over the GPIB is character data.

Character Data

Character data consists of ASCII characters grouped together in mnemonics that represent specific analyzer settings. The analyzer always returns the short form of the mnemonic in upper-case alpha characters. Character data looks like string data. Therefore, refer to the GPIB Command Finder to determine the return format for every command that can be queried.

Example of Character Data

```
MLOG
```

String Data

String data consists of ASCII characters. String parameters can contain virtually any set of ASCII characters. When sending string data to the analyzer, the string **must** begin with a single quote (') or a double quote (") and end with the same character (called the delimiter).

Note: The analyzer responds best to all special characters if the string is enclosed in single quotes. If quotes are not used, the analyzer will convert the text to uppercase. The analyzer may not respond as you expect.

The analyzer always encloses data in double quotes when it returns string data.

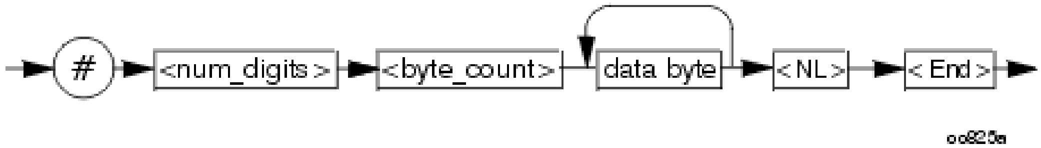
Example of String Data

GPIB.Write "DISP:WINDow:TITLe:DATA?"

```
"This is string response data."
```

Block Data

Block data is used to transfer measurement data. Although the analyzer will accept either definite length blocks or indefinite length blocks, it always returns definite length block data in response to queries unless the specified format is ASCII. The following graphic shows the syntax for definite block data:



<num_digits> specifies how many digits are contained in <byte_count>
<byte_count> specifies how many data bytes will follow in <data bytes>

Example of Definite Block Data

#17ABC+XYZ<nl><end>

- always sent before definite block data

1 - specifies that the byte count is one digit (7)

7 - specifies the number of data bytes that will follow, not counting <NL><END>

<NL><END> - always sent at the end of block data

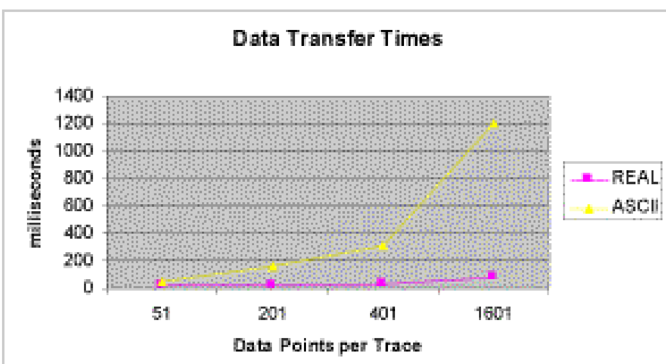
Transferring Measurement Data

Measurement data is blocks of numbers that result from an analyzer measurement. Measurement data is available from various processing arrays within the analyzer. For more information on the analyzer's data processing flow, see Accessing Data Map. Regardless of which measurement array is read, transferring measurement data is done the same.

When transferring measurement data, there are two data types to choose from:

- REAL
- ASCII

The following graphic shows the differences in transfer times between the two:



REAL Data

REAL data (also called floating-point data) types transfer faster. This is because REAL data is binary and takes about half the space of ASCII data. The disadvantage of using REAL data is that it requires a header that must be read. See definite length block data. The binary floating-point formats are defined in the IEEE 754-1985 standard. The following choices are available in REAL format:

- **REAL,32** - IEEE 32-bit format - single precision (not supported by HP BASIC)
- **REAL,64** - IEEE 64-bit format - double precision

These data types are selected using the FORMat:DATA command.

ASCII Data

The easiest and slowest way to transfer measurement data is to use ASCII data. If the data contains both numbers and characters, the setting of FORMat:DATA is ignored. ASCII data is separated by commas.

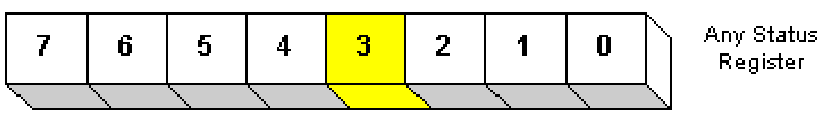
Reading the Analyzer's Status Register

The analyzer has several status registers that your program can read to know when specific events occur. There are two methods of reading the status registers in the analyzer: the Polled Bit method and the Service Request method.

- Polled Bit Method
- Service Request Method
- Setting and Reading Bits in Status Registers
- Positive and Negative Transitions
- Status Commands

Other Topics about GPIB Concepts

Most of the status registers in the analyzer are sixteen bits. For simplicity, this topic will illustrate their use with 8-bit registers. Bits in registers represent the status of a different conditions inside of the analyzer. In the following graphic, a register is represented by a row of boxes; each box represents a bit. Bit 3 is ON.



The Polled Bit Method

With the Polled Bit Method, your program **continually** monitors a bit in the status register that represents the condition of interest to you. When the analyzer sets the bit to 1, your program immediately sees it and responds accordingly.

Advantage: This method requires very little programming.

Disadvantage: This method renders your program unavailable to do anything other than poll the bit of interest until the condition occurs.

Procedure:

1. Decide which condition to monitor. The Status Commands topic lists all of the possible conditions that can be monitored in the analyzer.

2. Determine the command and the bit that will monitor the command.

3. Construct a loop to poll that bit until it is set.

4. Construct the routine to respond when the bit is set.

The Service Request (SRQ) Method

Your program enables the bits in the status registers representing the condition of interest. When the condition occurs, the analyzer actively interupts your program from whatever it is doing, and an event handler in your program responds accordingly. Do this method if you have several conditions you want to monitor or the conditions are such that it is not practical to wait for the condition to occur.

Advantage: This method frees your program to do other things until the condition occurs. The program is interupted to respond to the condition.

Disadvantage: This method can require extensive programming depending on the number and type of conditions that you want to monitor.

Procedure:

1. Decide which conditions to monitor. The Status Commands topic lists all of the possible analyzer conditions that can be monitored.

2. Set the **enable** bits in the **summary** registers and the **status byte** register.

> **Enabling** is like making power available to a light - without power available, the switch can be activated, but the light won't turn ON. In the analyzer, without enabling, the condition may occur, but the controller won't see it unless it is enabled.
>
> The condition, and the bit in the **summary** registers in the reporting path, must be enabled. Summary This is like streams (conditions) flowing into rivers (summary registers), and rivers flowing into the ocean (controller). See the diagram of status registers in Status Commands.
>
> Bit 6 of the **status byte** register is the only bit that can interrupt the controller. When **any** representative bit in the status byte register goes ON, bit 6 is automatically switched ON.

4. Enable your program to interupt the controller, This is done several ways depending on the programming language and GPIB interface card you use. An example program is provided showing how this is done with in Visual Basic with a National Instruments GPIB card.

5. Construct a subroutine to handle the interrupt event. If you are monitoring more than one condition in your system, your event handler must determine which condition caused the interupt. Use the *SPE command to determine the instrument that caused the interupt and then poll the summary registers, and then condition registers to determine the cause of the interupt.

Setting and Reading Bits in Status Registers

Both methods for reading status registers requires that you read bits out of the status registers. Most of the analyzers status registers contain 16 bits, numbered 0 to 15. Each bit has a weighted value. The following example shows how to set the bits in a 8-bit status register.

8-bit register

Bit	0	1	2	3	**4**	**5**	6	7
Weight	1	2	4	8	**16**	**32**	64	128

We want to set bits 4 and 5 in the Standard Event Status Enable register.

Step	Example
1. Read the weighted bit value for these bits	weights 16 and 32 (respectively)
2. Add these values together	16 + 32 = 48
3. Send this number as an argument in the appropriate command. (see Status Commands)	STAT:QUES:LIMIT1:ENAB 1026

Positive and Negative Transitions

Transition registers control what type of in a condition register will set the corresponding bit in the event register.

- **Positive** transitions (**0 to 1**) are only reported to the event register if the corresponding positive transition bit is set to 1.
- **Negative** transitions (**1 to 0**) are only reported to the event register if the corresponding negative transition bit is set to 1.
- Setting **both** transition bits to 1 causes both **positive and negative** transitions to be reported.

Transition registers are read-write and are unaffected by *CLS (clear status) or queries. They are reset to their default settings at power-up and after *RST and SYSTem:PRESet commands. The **following are the default settings** for the transition registers:

- All Positive Transition registers = 1
- All Negative Transition registers = 0

This means that by default, the analyzer will latch all event registers on the negative to positive transition (0 to 1).

The following is an example of why you would set transitions registers:

A critical measurement requires that you average 10 measurements and then restart averaging. You decide to poll the averaging bit. When averaging is complete, the bit makes a positive transition. After restart, you poll the bit to ensure that it is set back from 1 to 0, a negative transition. You set the negative transition bit for the averaging register.

Understanding Command Synchronization

The analyzer takes more time to process some commands than others:

- **Sequential** commands are processed quickly and in the order in which they are received.
- **Overlapped** commands take longer to process. Therefore, they allow the program to do other tasks while waiting. However, the programmer may want to prevent the analyzer from processing new commands until the overlapped command has completed. This is called "synchronizing" the analyzer and controller.

Note: The analyzer has two overlapped commands:
INITitate:IMMediate

SENSe:SWEep:MODE GROUPS (when INIT:CONT is ON)

The analyzer's queues store commands and responses waiting to be processed. Using the analyzer's queues and controlling the processing sequence of overlapped commands is called synchronizing the analyzer and the controller. This topic discusses how and when synchronizing should be performed.

- Analyzer Queues
- Synchronizing Overlapped Commands

Other Topics about GPIB Concepts

Analyzer Queues

Queues are memory buffers that store messages until they can be processed. The analyzer has the following queues:

- Input Queue

- Output Queue
- Error Queue

Input Queue

The controller sends statements to the analyzer without regard to the amount of time required to execute the statements. The input queue is very large (31k bytes). It temporarily stores commands and queries from the controller until they are read by the analyzer's command parser. The input queue is cleared when the analyzer is switched ON.

Output Queue

When the analyzer parses a query, the response is placed in the output queue until the controller reads it. Your program should immediately read the response or it may be cleared from the output queue. The following conditions will clear a query response:

- When a second query is sent before reading the response to the first. This does not apply when multiple queries are sent in the same statement.
- When a program statement is sent that exceeds the length of the input queue.
- When a response statement generates more data than fits in the output queue.
- When the analyzer is switched ON.

Error Queue

Each time the analyzer detects an error, it places a message in the error queue. When the `SYSTEM:ERROR?` query is sent, one message is moved from the error queue to the output queue so it can be read by the controller. Error messages are delivered to the output queue in the order they were received. The error queue is cleared when any of the following conditions occur:

- When the analyzer is switched ON.
- When the `*CLS` command is sent to the analyzer.
- When all of the errors are read.

If the error queue overflows, the last error is replaced with a `"Queue Overflow"` error. The oldest errors remain in the queue and the most recent error is discarded.

Synchronizing Overlapped Commands

GPIB commands are executed and processed by the analyzer in the order they are received. Commands can be divided into two broad classes:

- **Overlapped commands** generally take extended time to process by the analyzer. Examples of functions that have overlapped commands are printing and making measurements. Because they take longer to process, they allow the execution of subsequent commands while the overlapped command is still in progress. However, the programmer may want to prevent the analyzer from processing new commands until the overlapped command has completed. This is called "synchronizing" the analyzer and controller.
- **Sequential commands** are generally processed quickly by the analyzer. Therefore, they prevent the processing of subsequent commands until the sequential command has been completely processed. **These commands do NOT require synchronization.**

- Synchronization Methods
- When To Synchronize

Synchronization Methods

The following common commands are used to synchronize the analyzer and controller. Examples are included that illustrate the use of each command in a program. See the SCPI command details to determine if a command is an overlapped command.

- *WAI
- *OPC?
- *OPC

*WAI

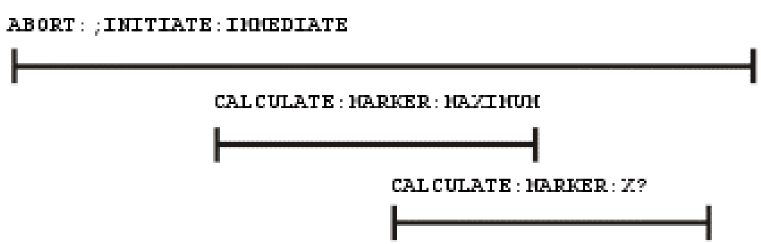
The *WAI command:

- **Stops the analyzer** from processing subsequent device commands until all overlapped commands are completed.
- **It does NOT stop the controller** from sending commands to this and other devices on the bus. This is the easiest method of synchronization.

Example of the *WAI command

```
GPIB.Write "ABORT;:INITIATE:IMMEDIATE" 'Restart the measurement.
 GPIB.Write "CALCULATE:MARKER:SEARCH:MAXIMUM" 'Search for max amplitude.
 GPIB.Write "CALCULATE:MARKER:X?" 'Which frequency?
```

The following timeline shows how the processing times of the three commands relate to each other:

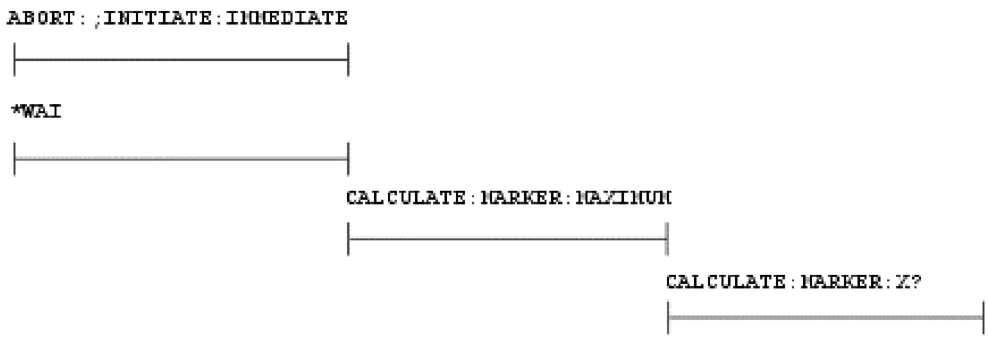


`INITIATE:IMMEDIATE` is an overlapped command; it allows the immediate processing of the sequential command, `CALCULATE:MARKER:SEARCH:MAXIMUM`. However, the `INITIATE:IMMEDIATE` is not considered complete until the measurement is complete. Therefore, the marker searches for maximum amplitude before the measurement completes. **The `CALCULATE:MARKER:X?` query could return an inaccurate value.**

To solve the problem, insert a `*WAI` command.

```
GPIB.Write "ABORT;:INITIATE:IMMEDIATE" 'Restart the measurement.
 GPIB.Write "*WAI" 'Wait until complete.
 GPIB.Write "CALCULATE:MARKER:MAXIMUM" 'Search for max amplitude.
 GPIB.Write "CALCULATE:MARKER:X?" 'Which frequency
```

The timeline now looks like this:



The `*WAI` command keeps the `MARKER:SEARCH:MAXIMUM` from taking place until the measurement is completed. The `CALCULATE:MARKER:X?` query returns the correct value.

Note: Although *WAI stops the analyzer from processing subsequent commands, it does not stop the controller. The controller could send commands to other devices on the bus.

***OPC?**

The *OPC? query **stops the controller until all pending overlapped commands are completed.**

In the following example, the **Read** statement following the *OPC? query will not complete until the analyzer responds, which will not happen until all pending overlapped commands have finished. Therefore, the analyzer and other devices receive no subsequent commands. A "**1**" is placed in the analyzer output queue when the analyzer completes processing an overlapped command. The "**1**" in the output queue satisfies the **Read** command and the program continues.

Example of the *OPC? query **Click**

This program determines which frequency contains the maximum amplitude.

```
GPIB.Write "ABORT; :INITIATE:IMMEDIATE"! Restart the measurement
GPIB.Write "*OPC?" 'Wait until complete
Meas_done = GPIB.Read 'Read output queue, throw away result
GPIB.Write "CALCULATE:MARKER:MAX" 'Search for max amplitude
GPIB.Write "CALCULATE:MARKER:X?" 'Which frequency?
Marker_x = GPIB.Read
PRINT "MARKER at " & Marker_x & " Hz"
```

*OPC

The *OPC command **allows the analyzer and the controller** to process commands while processing the overlapped command.

When the analyzer completes processing an overlapped command, the *OPC command sets bit 0 of the standard event register to 1 . This requires polling of status bytes or use of the service request (SRQ) capabilities of your controller. See Reading the Analyzer's Status Registers for more information about the standard event status register, generating SRQs, and handling interrupts.

Note: Be careful when sending commands to the analyzer between the time you send *OPC and the time you receive the interrupt. Some commands could jeopardize the integrity of your measurement. It also could affect how the instrument responds to the previously sent *OPC.

Example of polled bit and SRQ processes.

When To Synchronize the Analyzer and Controller

Although a command may be defined as an overlapped command, synchronization may not be required. The need to synchronize depends upon the situation in which the overlapped command is executed. The following section describes situations when synchronization is required to ensure a successful operation.

- Completion of a Measurement
- Measurements with External Trigger
- Averaged Measurements

Completion of a Measurement

To synchronize the analyzer and controller to the completion of a measurement, use the `ABORT;INITIATE:IMMEDIATE` command sequence to initiate the measurement.

This command sequence forces data collection to start (or restart) under the current measurement configuration. A restart sequence, such as `ABORT;INITIATE:IMMEDIATE` is an overlapped command. It is complete when all operations initiated by that restart command sequence, including the measurement, are finished. The `*WAI`, `*OPC?` and `*OPC` commands allow you to determine when a measurement is complete. This ensures that valid measurement data is available for further processing.

Measurements with External Trigger

To use an external trigger, synchronize the analyzer and controller before the trigger is supplied to the measurement. Setup the analyzer to receive a trigger from an external source (wired to the EXTERNAL TRIGGER connector on the rear panel. The trigger system is armed by GPIB with INITIATE:IMMEDIATE. (Because the source of the trigger has been specified as external, this command "readies" the analyzer for a trigger but it does not actually generate the trigger.).

Averaged Measurements

Averaged measurements are complete when the average count is reached. The average count is reached when the specified number of individual measurements is combined into one averaged measurement result. Use synchronization to determine when the average count has been reached.

If the analyzer continues to measure and average the results after the average count is reached, use synchronization to determine when each subsequent measurement is complete.

PNA as Controller and Controlled

The PNA does not have Pass control capability that other GPIB instruments have. Pass control allows the instrument to be programatically changed from being a controlled instrument to being the active controller ont he bus. However, there are other means for accomplishing the same thing. One is to control the PNA over LAN with VISA or SICL. See ... for more information on this.

The other way is to use a second GPIB port in the PNA. This can be done with a USB to GPIB interface card.

This is the hardware you need

This is how you configure it.

This is a sample program.

Configure for SCPI LAN using SICL / VISA

Programming the PNA using the SICL / VISA LAN Client interface to send and receive SCPI commands has several advantages over using the GPIB interface.

- No GPIB cables or interface card is necessary; the physical connection is over LAN
- The PNA can NOT be both a controller and talker/listener over GPIB at the same time. Using LAN to control the PNA leaves the PNA free to use the GPIB interface to control other GPIB devices.
- Data transfer speed is faster over LAN than GPIB

Note: SCPI commands can also be sent to the PNA using the SCPIStringParser of the COM interface. For optimum performance, use the COM interface to control the PNA objects directly.

To control the PNA using the SICL or VISA LAN Client interface, the external controller must have the Agilent I/O Libraries installed. Download a free copy at http://ftp.agilent.com/pub/mpusup/pc/binfiles/iop/index.html

The Agilent IO libraries include two libraries:

- VISA - the public-standard Virtual Instrument Software Architecture.
- SICL - the original Standard Instrument Control Library

Each of these libraries provides a software interface which will allow you to control your PNA with SCPI over LAN.

Note: The Agilent I/O Libraries are installed on the PNA. To run your SICL / VISA application on the PNA to control the PNA, set up a SICL or VISA LAN Client interface on the PNA, specifying the LAN hostname of that same PNA. This will work even if the PNA is not connected to a LAN.

Configure the PNA for SICL / VISA

1. On the PNA, click **System** then check **Windows Taskbar**
2. Click **Start** then point to **Program Files, Agilent IO Libraries**, then click **IO Config**
3. In the Configured Interfaces dialog box, click **hpib7** then click **Edit** (at the bottom of the dialog box). Note the **VISA Interface Name**.

4. Click **OK** to close the dialog, then click **OK** to close IO Config.
5. From the PNA **System** menu, point to Configure then click **SICL/GPIB**.
6. To enable SICL automatically when the PNA is rebooted, check **Automatically enable on Startup**. Otherwise, check **SICL Enabled** then click **OK.** Learn more about this dialog box.

The PNA is now ready to be controlled from within the PNA or over the LAN.

To Configure a PC to Control the PNA over LAN:

When configuring your controller PC, choose whether to use VISA or SICL. If you intend to have your code also support GPIB, then VISA is recommended as many different vendors of GPIB cards support VISA. SICL only supports Agilent GPIB cards.

1. On a PC with the Agilent I/O Libraries installed, click Start, then point to **Programs, Agilent IO Libraries**, then click **IO Config**. In the list of **Available Interface Types** click **LAN Client** , then click **Configure**.
2. In the **LAN Client** dialog box, click **OK**. In the Configured Interfaces box, you should see under **SICL Name** a new entry: lan or lanx, where x is an integer.
3. To use VISA,
    1. Click **VISA LAN Client** , then click **Configure**.
    2. In Remote Hostname, enter the full computer name of the PNA. Then click **OK**. Find your PNA computer name by going to Control Panel \ System \ Network Identification \ Full Computer name.
    3. In the I/O Config list of Configured Interfaces, you should see a new entry with VISA Name of GPIBx, where x is an integer.
4. Click **OK** to close I/O Config.
5. Use this example program to test your VISA configuration.

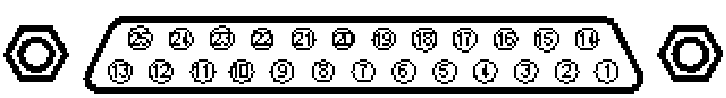
Other Topics about GPIB Concepts

Rear Panel Connectors

Auxiliary I/O Connector

General Description

This DB-25 male connector provides a variety of analog I/O, digital I/O, timing I/O, and supply lines. You can change the settings on the Auxiliary IO connector through SCPI and COM programming commands. The settings are NOT accessible through the front-panel keys or display menu.



Pin	Name	Description
1	ACOM	Ground reference for analog signals
2	Analog Out 2	-10 to +10Vdc output, 10mA max
3	Analog Out 1	-10 to +10Vdc output, 10mA max
4	no connect	for future enhancements
5	DCOM	Ground reference for digital signals

6	reserved	for future enhancements
7	reserved	for future enhancements
8	reserved	for future enhancements
9	+5V	+5Vdc output, 100mA max.
10	Pass/Fail Write Strobe	Indicates pass/fail line is valid (active low)
11	Sweep End	Indicates sweep is done (programmable modes)
12	Pass/Fail	Indicates pass/fail (programmable logic, modes and scope)
13	Output Port Write Strobe	Writes I/O port data (active low)
14	Analog In	-10 to +10VDC analog input
15	ACOM	Ground reference for analog signals
16	Power Button In	Grounding replicates front panel power button press
17	DCOM	Ground reference for digital signals
18	Ready for Trigger	Indicates ready for external trigger (active low)
19	External Trigger In	Measurement trigger input (programmable to be active high or low)
20	Footswitch In	Active low input latches a user-readable status bit.
21	+22V	+22Vdc output, 100mA max.
22	In/Out port C0	General purpose input / output
23	In/Out port C1	General purpose input / output
24	In/Out port C2	General purpose input / output
25	In/Out port C3_	General purpose input / output

ACOM (pins 1, 15)

Description

Analog common (ground) - To be used with the Analog Out and Analog In lines.

ACOM and DCOM are connected to system ground at a star ground point inside the analyzer.

Analog Out 1, 2 (pins 2, 3)

Description

Two analog outputs programmable to +/-10V; I_{out}<10mA; R_{out}=100 ohms

12-bit DACs with voltage resolution of approximately 5mV/count.

The DACs are set to constant values using SCPI or COM, and can be read using SCPI or COM commands.

Preset state for both pins is 0 volts.

HW Details

Looking into this output pin is a 100-ohm series resistor followed by two diodes tied to +/-15V for static protection, then the output or an op-amp.

The voltage output is provided by a 12-bit DAC with an op amp buffer.

Specifics:

- Maximum output current = 10mA
- Settling time = 3us

Timing

The DACs are set after the last data point is measured, during retrace. If the analyzer is in single sweep mode, the DACs are set as part of the presweep process, before the sweep is triggered.

DCOM (pins 5, 17)

Description

Digital common (ground).

Used with the digital input and output lines.

ACOM and DCOM are connected to system ground at a star ground point inside the analyzer.

Pins 6, 7, 8

Description

Reserved

+5V (pin 9)

Description

+5V nominal output (100mA max).

Protected by self-healing fuse:

Pass/Fail Write Strobe (pin 10)

Description

See Handler IO connector.

Sweep End (pin 11)

Description

See Handler IO connector.

Pass/Fail (pin 12)

Description

See Handler IO connector.

Output Port Write Strobe (pin 13)

Description

See Handler IO connector.

Analog In (pin 14)

Description

Analog input, +/-10V range, Rin=100k ohm

Bandwidth = 40kHz (2-pole lowpass filter).

This analog input may be read using the SCPI or COM commands.

HW Details

Looking into this pin there is 1k-ohm series resistor followed by 100k-ohm resistor to ground, static protection diodes after the 1k resistor limit the signal to +/-15V, then a high impedance buffer and active filter limiting the bandwidth to 40kHz with a lowpass filter.

Power Button In (pin 16)

Description

Short this pin to ground to replicate a front panel power button key press.

HW Details

Looking into the pin there is a 215-ohm series resistor followed by a 10k pull-up to the 3V standby supply, static protection diodes to the 0V/5V and then connects to the front panel power key circuit.

CAUTION: Because this line is internally pulled up to 3V, it should not be driven by a TTL driver.

Timing

Grounding this line for 1us to 2 seconds will simulate pressing the front panel power button.

Grounding this line for >4 seconds will perform a hard reset (similar to a personal computer) and is not recommended.

Ready for Trigger (pin 18)

Description

TTL output.

Active Low signal indicates that system is ready for an external trigger.

Remains High if system is not in External Trigger mode.

Goes High after an External Trigger is acknowledged.

Goes Low after the system has finished with its measurements, the source has been set up, and the next data point is ready to be measured.

HW Details

Looking into this pin there is a 215-ohm series resistor followed by a 10k pullup, diodes to 0V/5V for static protection, then the output of an "ABT" TTL buffer.

This line is enabled only when the analyzer is in External Trigger mode.

Refer to External Trigger In (following pin) for more information.

Timing

Refer to External Trigger In (following pin)

External Trigger In (pin 19)

Description

TTL input

This level-sensitive input will trigger the next measurement.

The trigger level mode is set by the user through the UI, SCPI or COM to either a TTL Low or a TTL High. Default is TTL High)

A single trigger is achieved by asserting the external trigger for a period from 1us to 50us. Continuous triggering is achieved by holding the external trigger in the "asserted" mode (either Low or High).

The External Trigger may trigger any of the following:

- next point measurement
- next channel measurement
- next Global measurement. (Default)

The External Trigger line is ignored if either "Ready For Trigger" is invalid or the analyzer is not in External Trigger mode. After a trigger, the analyzer will do the following:

- Autorange
- Measure data
- Move to the next measurement
- Indicate "ready for trigger".

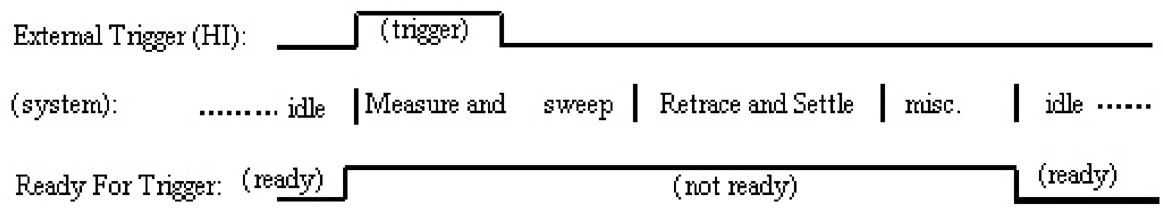
The preset state for Trigger is "Internal".

HW Details

Looking into this pin is a 215-ohm series resistor followed by a 4.64k pullup, 1000pF to ground and then a "FAST" TTL buffer input.

Timing

The trigger width should be between 1us and 50us.



Footswitch In (pin 20)

Description

TTL input.

A Low level input such as shorting this line to ground using a footswitch (where the input stays low for >1us) will be latched.

The latched status may be read using the SCPI or COM commands.

Only one footswitch press can be latched (remembered) by the system.

Reading the latch status will reset it if Footswitch In has returned to a high level.

HW Details

Looking into this pin is a 215-ohm series resistor followed by a 4.64k pullup to 5V and 1000pF to ground. This line is an input to a "FAST" TTL buffer.

Timing

Footswitch In must be Low for at least 1us.

+22V (pin 21)

Description

+22V nominal output (100mA max).
Protected by self-healing fuse.

In/Out Port C0-C3 (pins 22-25)

Description

See Handler IO connector

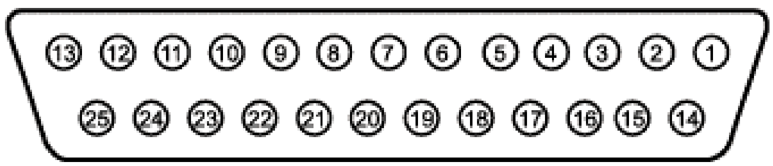
External Test Set I/O Connector

General Description

This DB-25 female connector is used to control external test sets. The external test set bus consists of 13 multiplexed address and data lines, three control lines, and an open-collector interrupt line. The Test Set IO is not compatible with the 8753 test sets.

You can change the settings on the External Test Set IO connector through SCPI and COM programming commands. The settings are NOT accessible through the front-panel keys or display menu.

Caution: Do not mistake this connector with a Parallel Printer port. A printer may be damaged if connected to this port.



Pin	Name	Description
1	SEL0	Test set select bit 0; tied to GND
2	Sweep Holdoff In	TTL input - state may be read with SCPI or COM command
3	AD12	Address and latched data
4	AD10	Address and latched data
5	AD9	Address and latched data
6	AD8	Address and latched data
7	GND	0V
8	LAS	TTL output Low = Address Strobe
9	AD4	Address and latched data
10	AD3	Address and latched data
11	AD2	Address and latched data
12	GND	0V
13	Interrupt In	TTL input - state may be read with a SCPI or COM command
14	No connect	**CAUTION**: Older PNAs have +22v on this line; this will damage a printer.
15	SEL1	Test set select bit 1; tied to GND
16	SEL2	Test set select bit 2; tied to GND
17	AD11	Address and latched data
18	SEL3	Test set select bit 3; tied to GND
19	AD7	Address and latched data
20	AD6	Address and latched data
21	AD5	Address and latched data
22	AD0	Address and latched data
23	AD1	Address and latched data
24	LDS	TTL output - active low data strobe
25	RLW	TTL output - high-read, low write

SEL0-SEL3 (pins 1,15,16,18)

Description

Selects addresses of test sets that are "daisy chained" to this port. The select code is set to zero at the PNA connector and is incremented by one as it goes through each successive external test set. Therefore, the first test set in the chain has address zero and so on, for up to 16 test sets.

HW Details

Connected to ground inside the PNA.

Timing

None

Sweep Holdoff In (pin 2)

Description

Input line used by the test set for holding off a sweep. Holding off a sweep is one way of introducing a delay that allows an external device to settle before the PNA starts taking data. You must write a program that will query the line and perform the delay. The program needs to query the line and keep PNA from sweeping while the line remains low. When a subsequent query detects that the line went high the program would then trigger the PNA to start the sweep.

Use either Single or External trigger mode to control the PNA sweep.

HW Details

This pin has a series 215-ohms resistor followed by 4.7k-ohm pull-up and then an "ABT" TTL buffered register.

Timing

This input is not latched by the PNA hardware. Therefore the input level must be held at the desired state by the test set until it's read by your program.

AD0-AD12 (pins 3-6, 9-11, 17, 19-23)

Description

Thirteen lines are used to output data addresses or input / output data. Several SCPI and COM commands are available for reading and writing to these lines. You can choose to use commands where the PNA provides the appropriate timing signals needed for strobing the addresses and data. Or you can choose to control the timing signal directly. The timing signals are RLW, LAS and LDS. If you decide to do direct control refer to the corresponding SCPI and COM command details. Close attention to detail is needed to insure the desired results.

After a write command, lines AD0-AD12 are left in the state they were programmed. Default setting for Mode is Read / Input).

After a read command, lines AD0-AD12 are left in input mode. While in this mode an external test set attached to the IO is free to set the level on each line.

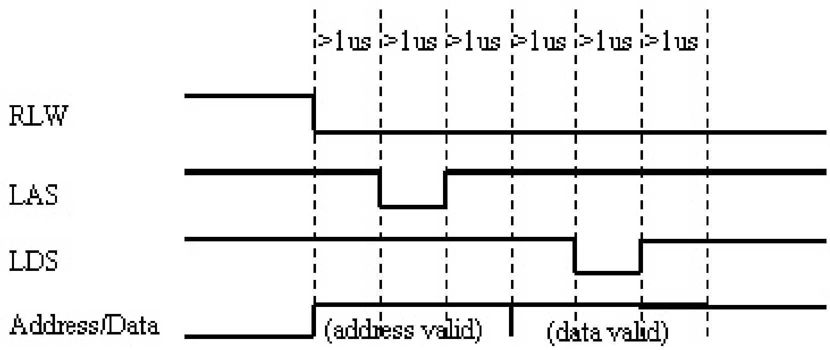
HW Details

Each of these I/O pins has a series 215-ohm resistor followed by 4.7k-ohm pull-up resistor.

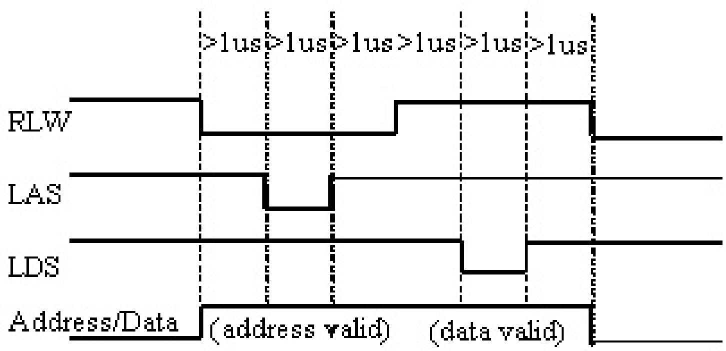
Write/Read is implemented by an output tri-state TTL buffer / latch for latching and enabling write data in parallel with a TTL input buffer for reading.

Timing

Output Address and data setup and hold times are 1us minimum.



Address & Data I/O Write



Address & Data I/O Read - Data must be valid for 1us before and after strobe

---

GND (pins 7, 12)

Description

Two ground pins used as ground references by the test set.

HW Details

Connected to digital ground.

Timing

None.

LAS (Low Address Strobe) (pin 8)

Description

This line has two behaviors that are command dependent. Refer to the SCPI and COM commands for further details.

In one behavior LAS is one of the lines used by the PNA to provide appropriate timing for writing Address and Data to the Test Set. In this case LAS is controlled automatically by the PNA and is intended to be used as the strobe for the Address. When LAS is low, lines AD0 - AD12 represent the Address. LAS will return to its normally high state when the transaction is finished.

In the second behavior the PNA will NOT provide appropriate timing. In this case LAS is controlled directly by the user through a SCPI or COM command. When the transaction is finished LAS is left set to the state it was programmed to until another command changes it. (Default for LAS is TTL High).

HW Details

This output pin is driven by a TTL latched buffer with a series 215-ohm resistor followed by 2.15k-ohm pull-up.

Timing

Strobe length, setup and hold times are all 1us minimum.

See the description for AD0-AD12 for more timing information.

Interrupt In (pin 13)

Description

Query this line with a SCPI or COM command.

HW Details

This line is a non-latched TTL input, has series 215-ohms followed by 4.64k-ohm pullup.

Timing

The Test Set must maintain at the desired TTL level until its read.

(pin 14) No Connect (previously +22V)

WARNING: Early versions of the PNA had +22v on this pin. **Connecting a printer to this port will usually damage the printer.**

Description

+22V, 100mA max. The 25-pin D connector is the same as a computer parallel printer port connector. Pin (14) corresponds to a printer's "autofeed" line. **Connecting a printer to this port will damage the printer if +22v is present** since printers requires less than 5V on all control lines.

HW Details

No connect

Timing

None

LDS (Low Data Strobe) (pin 24)

Description

This line has two behaviors that are command dependent. Refer to the External Test Set IO SCPI and COM commands for further details. (Default setting for LDS is TTL High)

In one behavior LDS is one of lines used by the PNA to provide appropriate timing for writing Address and Data to the Test Set. In this case LDS is controlled automatically by the PNA and is intended to be used as the strobe for the Data. When LDS is low, lines AD0 - AD12 represents Data. LDS will return to its normally high state when the transaction is finished.

In the second behavior the PNA will NOT provide appropriate timing. In this case LDS is controlled directly by the user through a SCPI or COM command. When the transaction is finished the LDS is left set to the state it was programmed to.

HW Details

This output pin is driven by a TTL latched buffer with a series 215-ohm resistor followed by 2.15k-ohm pull-up.

Timing

Strobe length, setup and hold times are all 1us minimum.

See the description for AD0-AD12 for more timing information.

RLW (pin 25)

Description

This line is the output for the Read Write signal. It has two behaviors that are command dependent. Refer to the External Test Set IO SCPI and COM commands for further details. (Default setting for RLW is TTL High)

In one behavior RWL is controlled automatically by the PNA during a Read Write operation. When RLW is low, lines AD0 - AD12 represent output Data. When RLW is high, the lines represent input Data.

In the second behavior the PNA does NOT provide the timing. The user must control it directly through the SCPI or COM command. In this case the line is left set to the state it was programmed to.

HW Details

This pin is a TTL latched output with a series 215-ohm resistor followed by 2.15k-ohm pull-up resistor.

Timing

Strobe length, setup and hold times are all 1us minimum.

See the description for AD0-AD12 for more timing information.

Material Handler I/O Connector

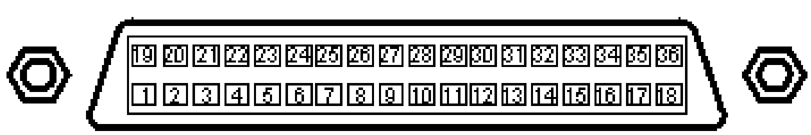
General Description

This rectangular 36-pin female connector provides four independent parallel data ports, nine control signal lines, one ground and a power supply line. All signals are TTL-compatible.

The data ports consist of two 8-bit output ports (Port A and Port B) and two 4-bit bidirectional ports (Port C and Port D).

You can change the settings on the Material Handler IO connector through SCPI and COM programming commands. The settings are NOT accessible through the front-panel keys or display menu.

See SCPI and COM Commands



There are two Handler IO pinout configurations: Type 1 and Type 2.

- **Type 1** - All RF PNA models (3 GHz, 6 GHz, and 9 GHz) are shipped from the factory with Type 1 pinout configuration. You can change the pinout configuration to Type 2 on these models. This requires opening the instrument and changing a connector internally.

Refer to the procedure in the Service Guide, Chapter 7. The Service Guide is available in .pdf format on a CD that was shipped with every PNA.
Caution: Changing this connection should be done by qualified service personnel.

- **Type 2** - All PNA models **EXCEPT** 3 GHz, 6 GHz, and 9 GHz are shipped with Type 2 configuration and cannot be changed.

Type 1 Handler IO pin assignments

Pin	Name	Description
1	Ground	0 V
2	INPUT1	TTL in, negative pulse (1us min) latches OUPUT1 & 2
3	OUTPUT1	TTL out, latched
4	OUTPUT2	TTL out, latched
5	Output port A0	TTL out, latched
6	Output port A1	TTL out, latched
7	Output port A2	TTL out, latched
8	Output port A3	TTL out, latched
9	Output port A4	TTL out, latched
10	Output port A5	TTL out, latched
11	Output port A6	TTL out, latched
12	Output port A7	TTL out, latched
13	Output port B0	TTL out, latched
14	Output port B1	TTL out, latched
15	Output port B2	TTL out, latched
16	Output port B3	TTL out, latched
17	Output port B4	TTL out, latched
18	no connect	
19	Output port B5	TTL out, latched
20	Output port B6	TTL out, latched
21	Output port B7	TTL out, latched
22	In/Out port C0	TTL in/out, latched
23	In/Out port C1	TTL in/out, latched
24	In/Out port C2	TTL in/out, latched
25	In/Out port C3	TTL in/out, latched
26	In/Out port D0	TTL in/out, latched
27	In/Out port D1	TTL in/out, latched
28	In/Out port D2	TTL in/out, latched
29	In/Out port D3	TTL in/out, latched
30	Port C Status	TTL out, Low= Input mode, High=Output mode
31	Port D Status	TTL out, Low= Input mode, High=Output mode
32	Output Port Write Strobe	TTL out, active Low data write strobe (1us min)
33	Pass/Fail	TTL out, latched, indicates pass/fail (programmable polarity)
34	Sweep End	TTL out, active Low (10us min) indicates sweep done
35	+5V	+ 5 V, 100mA max.
36	Pass/Fail Write Strobe	TTL out, active Low Pass/Fail write strobe (1us min)

Type 2 Handler IO pin assignments

Pin	Name	Description
1	Ground	0 V
2	INPUT1	TTL in, negative pulse (1us min) latches OUTPUT1 & 2
3	OUTPUT1	TTL out, latched
4	OUTPUT2	TTL out, latched
5	Output port A0	TTL out, latched
6	Output port A1	TTL out, latched
7	Output port A2	TTL out, latched
8	Output port A3	TTL out, latched

9	Output port A4	TTL out, latched
10	Output port A5	TTL out, latched
11	Output port A6	TTL out, latched
12	Output port A7	TTL out, latched
13	Output port B0	TTL out, latched
14	Output port B1	TTL out, latched
15	Output port B2	TTL out, latched
16	Output port B3	TTL out, latched
17	Output port B4	TTL out, latched
18	Output port B5	TTL out, latched
19	Output port B6	TTL out, latched
20	Output port B7	TTL out, latched
21	In/Out port C0	TTL in/out, latched
22	In/Out port C1	TTL in/out, latched
23	In/Out port C2	TTL in/out, latched
24	In/Out port C3	TTL in/out, latched
25	In/Out port D0	TTL in/out, latched
26	In/Out port D1	TTL in/out, latched
27	In/Out port D2	TTL in/out, latched
28	In/Out port D3	TTL in/out, latched
29	Port C Status	TTL out, Low= Input mode, High=Output mode
30	Port D Status	TTL out, Low= Input mode, High=Output mode
31	Output Port Write Strobe	TTL out, active Low data write strobe (1us min)
32	no connect	
33	Pass/Fail	TTL out, latched, indicates pass/fail (programmable polarity)
34	+5 V	+ 5 V, 100mA max.
35	Sweep End	TTL out, active Low (10us min) indicates sweep done
36	Pass/Fail Write Strobe	TTL out, active Low Pass/Fail write strobe (1us min)

Input1 (pin 2)

Description

A TTL input pulse is used to strobe user defined settings into the OUTPUT1 and OUTPUT2 lines. Latching occurs on the positive edge of INPUT1; minimum strobe length is 1us. Momentarily forcing this input Low, then High, will strobe the user data to the Output lines.

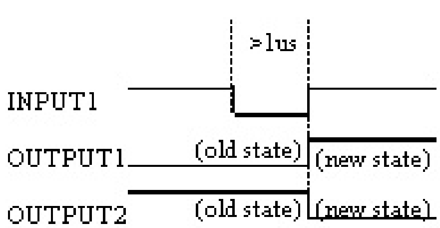
HW Details

This input has a 215-ohm series resistor followed by 10k-ohm pullup, a 1000pF capacitor to ground and a TTL buffer.

Timing

INPUT1 strobe length is 1us minimum.

OUTPUT1 and OUTPUT2 data is latched on the rising edge of INPUT1.



Output1, Output2 (pin3,4)

Description

The current state of these latched TTL outputs may be set High or Low (Default setting) using the SCPI or COM commands.

The next state (following a positive edge on the INPUT1 line) may be pre-loaded to High or Low (Default setting) using the commands.

HW Details

Looking back into these pins is a 215-ohm series resistor followed by 10k-ohm pullup, then the output of a TTL driver.

Timing

See INPUT1 timing.

Output port A0-A7, B0-B7

Description

Two general purpose 8-bit latched TTL output ports.

This data is valid when Output Write Strobe goes Low.

The preset state for data is TTL Low.

The logic of these ports may be defined as positive or negative (Default setting)

HW Details

Looking back into these pins is a 215-ohm series resistor followed by a 10k-ohm pullup.

These lines are driven by TTL general purpose latches.

Timing

Data has minimum 1us setup and hold times relative to the Data Write Strobe.

See Output Port Write Strobe for timing information.

In/Out port C0-C3, D0-D3

Description

Two general purpose 4-bit TTL input/output ports. Each port may be independently defined as either a 4-bit latched output port, or a 4-bit input port. The logic of these ports may be defined as positive or negative (Default setting). The logic setting cannot be independently assigned.

The four lines of Port C are connected internally to the Auxiliary IO connector Port C. The mode direction is not set automatically; it must be set by the user. The preset state for direction is "Input".

Setup and hold times of these lines relative to the Output Port Write Strobe are 1us.

HW Details

Looking back into pin, there is a 215-ohm series resistor followed by a 10k-ohm pullup. A diode is tied to +5V and ground for static protection.

These lines are driven by general purpose TTL latches and are read by general purpose TTL buffers.

The four lines of Port C are connected internally between the Handler IO and the Auxiliary IO connectors.

Timing

I/O Port output data is latched. Relative to the I/O Port Write Out strobe, the setup and hold times are guaranteed to be a minimum of 1us. See Output Port Write Strobe for timing information

---

Port C Status, Port D status

Description

Latched TTL outputs indicate direction of the C and D ports. A Low level on the status line indicates that the associated port is in the **INPUT** mode (read only).

A High level indicates the associated port is in **OUTPUT** mode (write only). These outputs are not affected by the logic of the ports.

The status lines are set when the command that sets the port mode is sent.

HW Details

Looking back into these pins, there is a 215-ohm series resistor followed by a 10k-ohm pullup.

These lines are driven by general purpose TTL latches.

Timing

None.

Output Port Write Strobe

Description

Normally High, this TTL output goes Low (for minimum of 1us) to write data from the two 8-bit and two 4-bit data ports on the Handler IO and In/Out Port C on the Auxiliary IO port. This line is not affected by the port logic.

HW Details

Looking back into the pin is a series 215-ohm resistor followed by 10k-ohm pullup.

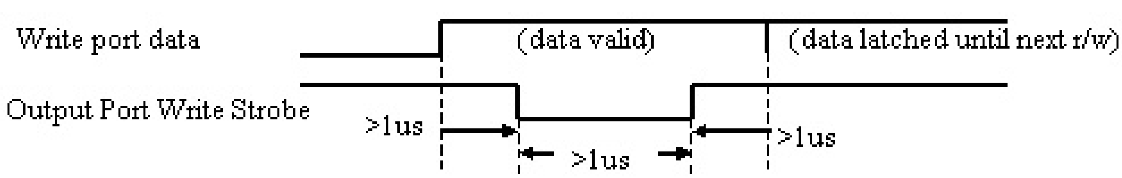
Connected to a TTL register output controlled by the analyzer.

Shared between the Handler IO and the Auxiliary IO.

Timing

Active low strobe; low for a minimum 1us.

Setup and Hold times relative to the I/O Port data lines are 1us minimum.



Pass/Fail (pin 33)

Description

Latched TTL output indicates whether the limit test has passed or failed. The Pass/Fail line is valid when Pass/Fail strobe line is active low.

The logic levels may be set to the following using SCPI or COM commands:

- Positive Logic: High=Pass, Low=Fail. (Default setting)
- Negative Logic: High=Fail, Low=Pass.

The default state of the line may be set to the following using SCPI or COM commands:

- **Default Pass No Wait mode:** Pass/Fail line indicates a pass until a failure is detected, at which time the output immediately indicates a failure. Pass/Fail line resets to "pass" when the source has been reset and the receiver is ready to take new data. (Default setting)
- **Default Pass Wait mode:** Pass/Fail line indicates a pass until the measurement has finished and all limits have been tested, at which time the output will indicate whether a

fail was detected. The Pass/Fail line is reset to "pass" when the source has been reset and the receiver is ready to take new data.

- **Default Fail mode:** Pass/Fail line indicates a failure until the measurement has finished and all limits have been tested, at which time the output will indicate whether a pass was detected. The Pass/Fail line resets to "fail" when the source has been reset and the receiver is ready to take new data.

The scope of the line may be set to the following using SCPI or COM commands:

- **Channel scope:** Pass/Fail line will have channel scope. The line resets to the default state after the measurements on a channel have completed.
- **Global scope:** Pass/Fail line will have Global scope. The line resets to the default state after the measurements on all triggerable channels have completed. (Default setting)

Pass/Fail output is active only when the limit test function is on. It is set to indicate a the default condition when the limit test function is off.

HW Details

This line is shared between the Handler IO and the Auxiliary IO connector.

Looking into this pin there is a series 215-ohm resistor followed by a 10k pullup and is driven by a TTL register.

Timing

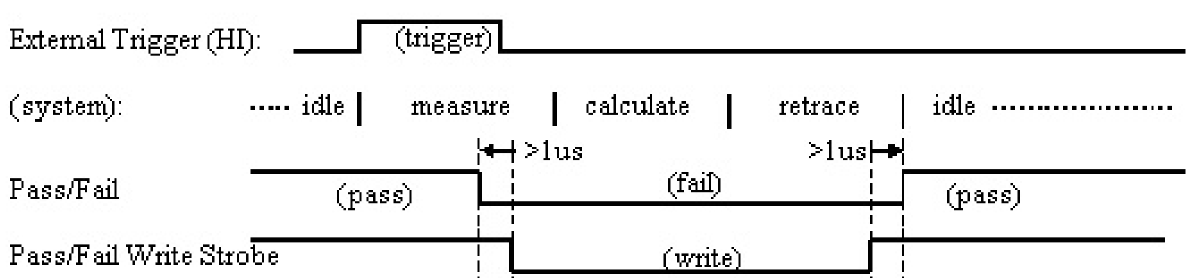
The Pass/Fail Out state is valid for at least 1us before Pass/Fail Write Strobe is pulled Low.

The Pass/Fail Out state is valid for at least 1us after Pass/Fail Write Strobe is pulled High.

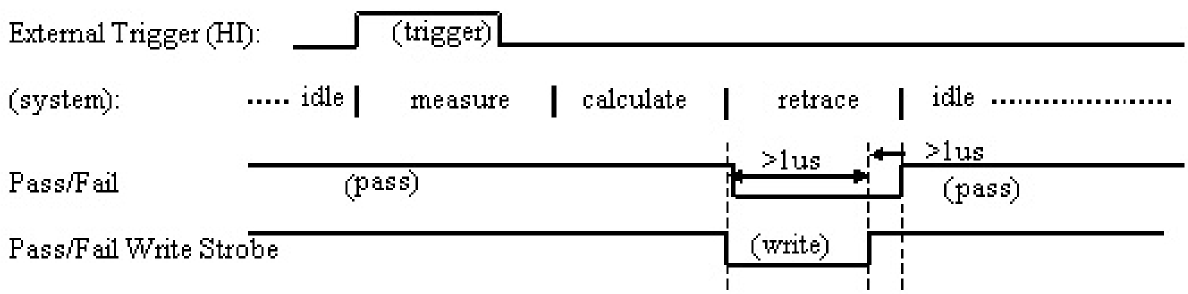
Pass/Fail Out is reset to its default state before the next measurement is started.

Pass/Fail Write Strobe will be Low for at least 1us.

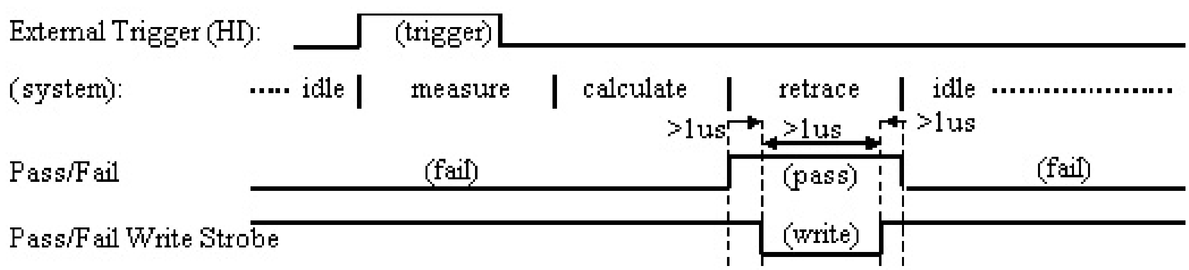
If the network analyzer is in **External Trigger** mode, Pass/Fail Write Strobe will go High (invalid) at least 1us before **Ready for Trigger** goes Low.



Pass/Fail (default "pass" mode, positive logic, no wait mode)



Pass/Fail (default "pass" mode, positive logic, end-of-measurement mode)



Pass/Fail (default "fail" mode, positive logic)

+5V

Description

+5V nominal output (100mA max).

Protected by self-healing fuse.

Sweep End

Description

Low TTL output (10us minimum) indicates that the specified sweep event has finished. High output (10us minimum) indicates that the specified sweep event is active. The sweep event includes sweeping the source and taking data.

The Sweep Event Mode may be set to the following using SCPI and COM commands:

- **Sweep**: indicates that a single source sweep has finished. (Default setting)
- **Channel**: indicates that a single channel has finished.
- **Global**: indicates that all enabled channels have finished.

HW Details

Looking into this pin, there is a 215-ohm series resistor followed by a 10k-ohm pullup. This line is driven by a TTL register.

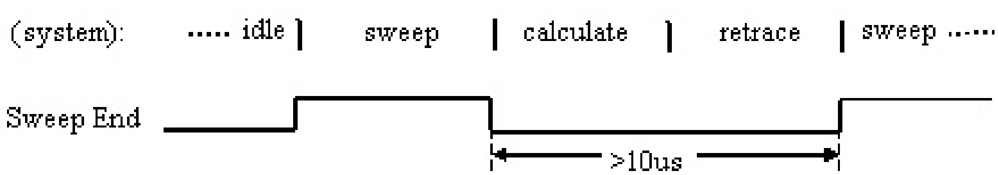
This line is shared between the Handler IO and the Auxiliary IO connectors.

Timing

Sweep End Out is guaranteed to be High while the sweep event is active. Its falling edge indicates that the sweep event has finished and is usually low while the sweep event is inactive.

Sweep End Out is guaranteed to be Low for a minimum of 10us and High for a minimum of 10us.

Note: Sweep End = Low does not indicate that all calculations have finished.



Pass/Fail Write Strobe (pin 36)

Description

Active low TTL output strobe indicates that "Pass/Fail Out" is valid.

Relative to the "Pass/Fail Out" line, this strobe has a minimum setup, strobe length, and hold time of 1us each.

The Pass/Fail Strobe is fixed in duration and timing. However, the occurrence of the strobe depends on the Pass/Fail Mode and Pass/Fail Scope (Channel or Global) settings.

The Pass/Fail mode may be set to the following using SCPI and COM commands:

PASS- the line stays in PASS state. When a device fails, then the line goes to fail after the Sweep End line is asserted.

FAIL- the line stays in FAIL state. When a device passes, then the line goes to PASS state after the Sweep End line is asserted.

No Wait- the line stays in PASS state. When a device fails, then the line goes to fail IMMEDIATELY.

HW Details

This line is shared between the Handler IO and the Auxiliary IO connectors.

Looking into this pin, there is a 215-ohm series resistor followed by a 10k pullup. This line is driven by TTL logic.

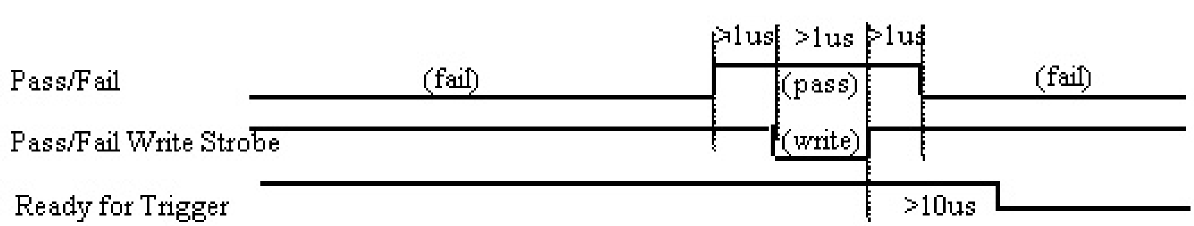
Timing

The Pass/Fail Out state will be valid for at least 1us before Pass/Fail Write Strobe is pulled Low (which indicates that Pass/Fail Out is valid).

The Pass/Fail Out state will be valid for at least 1us after Pass/Fail Write Strobe is pulled High.

Pass/Fail Write Strobe will be Low for at least 1us.

If the network analyzer is in "External Trigger" mode, Pass/Fail Write Strobe will go High (invalid) at least 10us before "Ready for Trigger" goes Low.



See Pass/Fail output for more timing information.

8753 Command Cross Reference

Symbol Conventions

Symbol	Description
<num>	Required numerical data.
<a1| a2>	An appendage that is part of the command. For example, FORMAT<DOS|LIF> indicates that the actual commands are FORMATDOS and FORMATLIF.
<$>	Indicates a character string operand which must be enclosed by double quotes.
|	An either/or choice in appendages or optional data.
[ ]	Optional data.
<LF>	Line feed.

Description of Symbol Conventions

Legend

Indicates the most common of the network analyzer commands that have been mapped to a corresponding command in PNA. Since the commands listed on this page are base commands, commands that are derived from these base commands may not have a corresponding command in PNA.

Indicates a command that has **not** been mapped to a corresponding command in PNA, but may be in future revisions. However, this does not always indicate that the required functionality does not exist in PNA. See the 8753 Programming Guide for a description of the command functionality.

AB

ADAP1

ADDR

ADPT

ALC

ALTAB

ANAB

ANAI

AR

ASEG

ASSS

ATT

AUTO

AUXC
AVER
BACI
BANDPASS
BEEP
BLAD
BR
CAL1
CALF
CALI
CALK
CALN
CALSPORT
CALZ
CBRI
CENT
CHAN
CHOPAB
CLAD
CLASS
CLEA
CLEAL
CLEABIT
CLEASEQ
CLEL
CLES
CLS
COAD
COAX
COLO
COLOR
CONS
CONT
CONV
COPY
CORI
CORR
COU
CSWI
CWFREQ
CWTIME
D1DIVI2
D2XUPCH

- D4XUPCH
- DATI
- DCONV
- DEBU
- DECRLOOC
- DEFC
- DEFLPRINT
- DEFLTCPIO
- DEFS
- DEL
- DELA
- DEMO
- DFLT
- DIRS
- DISC
- DISM
- DISP
- DIVI
- DONE
- DONM
- DOSEQ
- DOWN
- DUAC
- DUPLSEQ
- ECALAB?
- ECALCONT
- ECALDONE
- ECALERC
- ECALFREQS
- ECALFUL2
- ECALISOAVG
- ECALMANTHRU
- ECALMODID
- ECALMODINF
- ECALMODSELA
- ECALMODSELB
- ECALNFREQS
- ECALOMII
- ECALPAUSED
- ECALRERC
- ECALS11
- ECALS22
- EDIT

ELED
EMIB
ENTO
ERCDONE
ESB?
ESE
ESNB
ESR?
EXTD
EXTM
EXTRCHAN
EXTT
FIXE
FORM
FORMAT
FREQ
FREQOFFS
FRER
FLUP
FWD
GATE
GATS
GOSUB
HARM
HOLD
IDN?
IF
IFBW
IMAG
INCRLOOC
INI
INPU
INSM
INT
INTE
ISO
KEY
KITD
KOR?
LAB
LABE
LEF
LIM

LIMI
LIMT
LINFREQ
LINM
LINT
LIS
LISTTYPE
LISV
LO
LOA
LOAD
LOADSEQ
LOGFREQ
LOGM
LOOC
LOWP
LRN
MANTRIG
MARK
MAXF
MEAS
MEASTAT
MENU
MINF
MINMAX
MINU
MODI1
MODS
NEWSEQ
NEXP
NOOP
NUMG
NUMR
OF
OFS
OMII
OPC
OPEP
OUTP
PARA
PARAL
PAUS
PCB

PCOL
PENN
PHAO
PHAS
PLOS
PLOT
PLT
PMTRTTIT
POIN
POL
PORE
PORT
PORTP
POWE
POWL
POWM
POWR
POWS
POWT
PRAN
PREP
PRES
PRI
PRIN
PRINTALL
PRN
PTOS
PURG
PWMC
PWRLOSS
PWRMCAL
PWRR
RAI
RAWOFFS
READ
REAL
RECA
RECO
REF
REFT
REIC
RERCDONE
RESC

RESD
RESPDONE
REST
RETP
REV
RF
RFLP
RIG
RSCO
RST
S
SADD
SAMC
SAV
SAVE
SAVEUSEK
SAVU
SCAL
SCAP
SDEL
SDON
SEA
SEDI
SEG
SEL
SELL
SEQ
SEQWAIT
SET
SHOM
SING
SLI
SLOP
SM8
SMI
SMOO
SOFR
SOFT
SOUP
SPAN
SPEC
SPEG
SPLD

SPLID
SRE
SSEG
STAN
STAR
STB?
STDD
STDT
STEPSWP
STOP
STOR
STORSEQ
STPSIZE
SVCO
SWE
SWPSTART
SWR
TAK
TAKE4
TALKLIST
TESS?
TIMDTRAN
TIMESTAM
TINT
TIT
TITT
TRA
TRACK
TRAP
TRL
TSSWI
TST?
TSTIO
TSTP
TTL
UCONV
UP
USEPASC
USESENS
VELOFACT
VIEM
VOFF
WAIT

- WAVD
- WAVE
- WID
- WIND
- WRSK

AB

8753 Command	Description	Range	Query Response
AB	Measures and displays A/B on the active channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

Step 1 CALC:PAR:DEFINE	Create the measurement.
Step 2 DISP:WIND ON	If a new window will be used to display the measurement, then create a window.
Step 3 DISP:WIND:TRAC:FEED	Display the measurement in the window.

PNA COM Equivalent - Notes

CreateMeasurement Method	Create and display the measurement.

ADDR

8753 Command	Description	Range	Query Response
ADDRPOWM	Power Meter GPIB address	Integers 0-30	<num>><LF>

PNA SCPI Equivalent - Notes

SYST:COMM:GPIB:PMET:ADDR	Specifies the GPIB address of the power meter to be used in a source power calibration.

PNA COM Equivalent - Notes

PowerMeterGPIBAddress Property	Specifies the GPIB address of the power meter that will be referenced by the SourcePowerCalibrator object

---

ALTAB

8753 Command	Description	Range	Query Response
ALTAB	Places the analyzer in the alternate inputs measurement mode, where A and B measurements are made on alternate sweeps. See also "CHOPAB."	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:COUP ALL	Sets sweeps to either alternate or chopped.

PNA COM Equivalent - Notes

AlternateSweep Property	Sets sweeps to either alternate or chopped.

ANAI

8753 Command	Description	Range	Query Response
ANAI	Measures and displays the data at the Auxiliary Input	Integers 1-31	<0|1>><LF

(Analog IN

PNA SCPI Equivalent - Notes

CONT:AUX:INP	Reads the ADC input voltage from pin 14 of the AUX IO connector.

PNA COM Equivalent - Notes

get InputVoltage Method	Reads the ADC input voltage from pin 14 of the AUX IO connector.

AR

8753 Command	Description	Range	Query Response
AR	Measures and displays A/R on the active channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

Step 1 CALC:PAR:DEFINE	Create the measurement.
Step 2 DISP:WIND	If a new window will be used to display the measurement, then create a window.
Step 3 DISP:WIND:TRAC:FEED	Display the measurement in the window.

PNA COM Equivalent - Notes

CreateMeasurement Method	Create and display the measurement.

ASEG

8753 Command	Description	Range	Query Response
ASEG	Uses all segments for list frequency sweep. See also "SSEG".	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:SEGM	Turn on each segment to be used with list frequency sweep.

PNA COM Equivalent - Notes

LimitSegment Object	LimitSegment object.

ATT

8753 Command	Description	Range	Query Response
ATTP1><num>[DB]	Selects the amount of attenuation at **PORT 1**.	0–70 dB	<num><LF
ATTP2><num>[DB]	Selects the amount of attenuation at **PORT 2**. Note: These commands only apply to 8753ES Option 011 analyzers.	0–70 dB	<num><LF

PNA SCPI Equivalent - Notes

Step 1 SOUR:POW:COUP	Set Port Power Coupling OFF.
Step 2 SOUR:POW:ATT	Set the attenuation level for the selected port.

PNA COM Equivalent - Notes

Step 1 Couple Ports Property	Set Port Power Coupling OFF.
Step 2 Attenuator Property	Set the attenuation level for the selected port.

AUTO

8753 Command	Description	Range	Query Response

AUTO	Auto scale the active channel.	N/A	N/A

PNA SCPI Equivalent - Notes

DISP:WIND:TRAC:Y:AUTO	Auto scale on the specified trace in the specified window.

PNA COM Equivalent - Notes

Autoscale Method	Auto scales the trace or all of the traces in the selected window.

AVER

8753 Command	Description	Range	Query Response
AVERREST	Restarts the averaging on the active channel.	N/A	N/A
AVERFACT<num>	Sets the averaging factor on the active channel.	integers 0–999	<num><LF
AVERO<ON|OFF>	Turns averaging on and off on the active channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:AVER:CLE	Restart averaging.
SENS:AVER:COUN	Read-Write the averaging factor.
SENS:AVER	Read-Write averaging ON or OFF.

PNA COM Equivalent - Notes

Averaging Restart Method	Restart averaging.
Averaging Factor Property	Read-Write the averaging factor.
Averaging Property	Read-Write averaging ON or OFF.

BLAD

8753 Command	Description	Range	Query Response
BLAD<ON|OFF>	Blanks the display.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

DISP:ENAB	Blanks the display information in **all** windows.
DISP:WIND:ENABle	Blanks the display information in **a specified** window.

PNA COM Equivalent - Notes

Visible Property	Makes the Network Analyzer application visible or not visible.

BR

8753 Command	Description	Range	Query Response
BR	Measures and displays B/R on the active channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

	Follow the steps below to create and display a measurement.
Step 1 CALC:PAR:DEF	Create the measurement.
Step 2 DISP:WIND	If a new window will be used to display the measurement, then create a window.
Step 3 DISP:WIND:TRAC:FEED	Display the measurement in the window.
PNA COM Equivalent - Notes	
CreateMeasurementMethod	Create and display the measurement.

CALF

8753 Command	Description	Range	Query Response
CALFCALF	Sets the power meter sensor calibration factor.	0200%	<num><L F >
CALFSEN<A|B>	Edits a apecified power sensor calibration table	<N/A>	<N/A>

PNA SCPI Equivalent - Notes	
SOUR:POW:CORR:COLL:TABL:DATA	(Read-Write) Read or write data into the selected table. If the selected table is a power sensor table, the data is interpreted as cal factors in units of percent. If the loss table is selected, the data is interpreted as loss in units of dB.
SOUR:POW:CORR:COLL:TABL	Selects which table (cal factor table for a power sensor, or the loss compensation table) you want to write to or read from.
PNA COM Equivalent - Notes	
CalFactor Property	Sets or returns the cal factor value associated with a power sensor cal factor segment.
CalFactorSegments Collection PowerLossSegments Collection	Access the appropriate table in either collection.

CALI

- CALIERC
- CALIRERC
- CALIFUL2
- CALIRAI
- CALIRESP
- CALIS111
- CALIS221
- CALITRL2

8753 Command	Description	Range	Query Response
CALIFUL2	Begins the sequence for a short, load, open, thru (SLOT) 2-port calibration.	N/A	<0|1><LF>
CALIRAI	Begins the sequence for a response and	N/A	<0|1><LF>

	isolation calibration.		
CALIRESP	Begins the sequence for a response calibration.	N/A	<0|1><LF>
CALIS111	Begins the sequence for an S11 1-port calibration (ES models), or a reflection 1-port calibration (ET models).	N/A	<0|1><LF>
CALIS221	Begins the sequence for an S22 1-port calibration.	N/A	<0|1><LF>
CALITRL2	Begins the sequence for a thru, reflect, line *or* line, reflect, match (TRL*/LRM*) 2-port calibration.	N/A	<0|1><LF>

PNA SCPI Equivalent - Notes

SENS:CORR:COLL:CKIT	If a calibration kit is not selected, select a calibration kit.
SENS:CORR:COLL	Measure the specified standard from the selected calibration kit.

PNA COM Equivalent - Notes

CalKitType Property	If a calibration kit is not selected, select a calibration kit.
AcquireCalStandard2 Method	Measure the specified standard from the selected calibration kit.

8753 Command	Description	Range	Query Response
CALIERC	Begins the sequence for a **forward** enhanced response calibration.	N/A	<0|1><LF>
CALIRERC	Begins the sequence for a **reverse** enhanced response calibration.	N/A	<0|1><LF>

Notes

These commands currently are not available.

CALK

8753 Command	Description	Range	Query Response
CALK24MM	Selects a 2.4-mm calibration kit (85056A/D) as the default cal kit.	N/A	<0|1>><LF
CALK292MM	Selects a 2.92-mm calibration kit as the default cal kit.	N/A	<0|1>><LF
CALK292S	Selects a 2.92* calibration kit	N/A	<0|1>><LF

<table>
<tr><td></td><td>(85056K) as the default cal kit.</td><td></td><td></td></tr>
<tr><td>CALK35MD</td><td>Selects a 3.5-mm calibration kit (85052B/D for 8720E series analyzers, and 85033D for 8753ET/ES analyzers) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALK35MC</td><td>Selects a 3.5-mm calibration kit (85033C) as the default cal kit. CALK35MM selects the 85033C cal kit for the 8752C and 8753D analyzers.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALK716</td><td>Selects a 7-16 calibration kit (85038) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALK7MM</td><td>Selects a 7-mm calibration kit (85050 series for 8720E series analyzers, and 85031B for 8753ET/ES analyzers) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALKN50</td><td>Selects a type-N 50 ohm calibration kit (85054 for 8720E series analyzers, and 85032B/E for 8753ET/ES analyzers) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALKN75</td><td>Selects a type-N 75 ohm calibration kit (85036B/E) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALKTRLK</td><td>Selects a TRL 3.5-mm calibration kit (85052C) as the default cal kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>CALKUSED</td><td>Selects a user-defined calibration kit.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes	
SENS:CORR:COLL:CKIT	Select the appropriate calibration kit. If the specific calibration kit is not listed, one can be created and stored in the User Defined section. 3 = CALKN50 4 = CALK35MD 5 = CALK716

		6 = CALKTRLK	

PNA COM Equivalent - Notes

CalKitType Property	Specifies the type of calibration kit to use in the calibration process.

CALN

8753 Command	Description	Range	Query Response
CALN	Turns calibration type to "off." See also "CORR."	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:CORR:COLL:METH REFL1	Read-Write the calibration method. Set to "NONE" to turn calibration off.

PNA COM Equivalent - Notes

CalibrationType_Property	Specifies and returns the type of calibration to be **applied** to the measurement. Set to "NONE" to turn calibration off.

CENT

8753 Command	Description	Range	Query Response
CENT<num>[HZ|DB]	Sets the center stimulus value. If a list frequency segment is being edited, sets the center of the list segment.	For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ± 1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes

SENS:FREQ:CENT	To set the center frequency value.
SENS:SEGM:FREQ:CENT	To set the center frequency value **for a segment**.

PNA COM Equivalent - Notes

CenterFreq_Property	Read-Write the center frequency of all measurements in a channel or for a specified sweep segment.

CHAN

8753 Command	Description	Range	Query Response
CHAN1	Makes channel 1 the active channel. OPC-compatible.	N/A	N/A
CHAN2	Makes channel 2 the active channel. OPC-compatible.	N/A	N/A
CHAN3	Makes channel 3 the active channel. OPC-compatible.	N/A	N/A

CHAN4	Makes channel 4 the active channel. OPC-compatible.	N/A	N/A

Notes

Unlike the 8753 network analyzer, the PNA Series Network Analyzers do not need a separate channel to display each parameter. While the PNA Series has four independent measurement channels, only one channel is needed to display all four measurement parameters. In addition, up to four windows are available to view four active and four memory traces per window.

CHOPAB

8753 Command	Description	Range	Query Response
CHOPAB	Places the analyzer in the chop measurement mode. See also "ALTAB."	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:COUP ALL	Read-Write the sweep mode as Chopped or Alternate.

PNA COM Equivalent - Notes

Alternate_Sweep_Property	Read-Write the sweep mode as Chopped or Alternate.

CLASS

8753 Command	Description	Range	Query Response
CLASS11A	S11A: S11 (forward reflection) 1-port, open	N/A	N/A
CLASS11B	S11B: S11 (forward reflection) 1-port, short	N/A	N/A
CLASS11C	S11C: S11 (forward reflection) 1-port, load	N/A	N/A
CLASS22A	S22A: S22 (reverse reflection) 1-port, open	N/A	N/A
CLASS22B	S22B: S22 (reverse reflection) 1-port, short	N/A	N/A
CLASS22C	S22C: S22 (reverse reflection) 1-port, load	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:CORR:COLL	Measure the specified standard from the selected calibration kit.

PNA COM Equivalent - Notes

AcquireCalStandard2_Method	Measure the specified standard from the selected calibration kit.

CLEAL

8753 Command	Description	Range	Query Response
CLEAL	Clears the limit line list. Should be preceded by EDITLIML.	N/A	N/A

PNA SCPI Equivalent - Notes

CALC:LIM:DATA	Limit lines always remain in memory. Use this SCPI command to set limit segment OFF or make a new limit line.

PNA COM Equivalent - Notes

Delete_Method	Delete the limit test collection.

CLEL

8753 Command	Description	Range	Query Response
CLEL	Clears the currently selected list. This could be a frequency list, power loss list, or limit test list. Must be preceded by an "EDIT" command.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:SEGM:DEL	Clear a **single** sweep segment.
SENS:SEGM:DEL:ALL	Clear **all** sweep segments.

PNA COM Equivalent - Notes

Remove_Method	Removes an item from a collection of objects.

CLES

8753 Command	Description	Range	Query Response
CLES	Clears the status byte register, the event-status registers, and the enable registers. Same as CLS.	N/A	N/A

PNA SCPI Equivalent - Notes

*CLS - Clear Status	Clears the instrument status byte by emptying the error queue and clearing all event registers.

PNA COM Equivalent - Notes

	No equivalent command at present.

CLS

8753 Command	Description	Range	Query Response
CLS	Clears the status byte register, the event-status registers, and the enable registers. Same as CLES.	N/A	N/A

PNA SCPI Equivalent - Notes

*CLS - Clear Status	Clears the instrument status byte by emptying the error queue and clearing all event registers. Replace "CLS" with "*CLS".

PNA COM Equivalent - Notes

No equivalent command at present.

CONT

8753 Command	Description	Range	Query Response
CONT	Places the analyzer in continuous sweep trigger mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
INIT:CONT	Read-Write the sweep triggering mode.

PNA COM Equivalent - Notes	
Continuous_Method	Read-Write the sweep triggering mode.

CORI

8753 Command	Description	Range	Query Response
CORI<ON|OFF>	Turns interpolative error correction on and off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:CORR:INT	Read-Write correction interpolation ON or OFF.

PNA COM Equivalent - Notes

No equivalent command at present.

CORR

8753 Command	Description	Range	Query Response
CORR<ON|OFF>	Turns error correction on and off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:CORR	Read-Write whether or not correction data is applied to the measurement.

PNA COM Equivalent - Notes	
Error_Correction_Property	Sets (or returns) error correction ON or OFF

CWFREQ

8753 Command	Description	Range	Query Response
CWFREQ<num>[HZ|DB]	Sets the CW frequency for power sweep and CW frequency modes. While the list frequency table segment is being edited, it sets the center frequency of the current segment. See also "MARKCENT."	For frequency or power sweeps, refer to "Preset State and Memory Allocation," in the analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ±1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes	
SENS:FREQ	Read-Write the Continuous Wave (or Fixed) frequency.

PNA COM Equivalent - Notes	
CW_Frequency_Property	CW Frequency property.

CWTIME

8753 Command	Description	Range	Query Response
CWTIME	Selects CW time as the sweep type.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:SWE:TYPE	Read-Write the type of analyzer sweep mode.

PNA COM Equivalent - Notes	
Sweep_Type_Property	Sets the type of X-axis sweep that is performed on a channel.

DATI

8753 Command	Description	Range	Query Response
DATI	Stores the data trace in channel memory. OPC-compatible.	N/A	N/A

PNA SCPI Equivalent - Notes	
CALC:MATH MEM	Write-only the currently selected measurement trace into memory.

PNA COM Equivalent - Notes	
DataToMemory_Method	Stores the active measurement into memory.

DEL

8753 Command	Description	Range	Query Response
DELO	Turns delta marker mode off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:MARK:DELT	Read-Write whether marker is relative to the reference or not.

PNA COM Equivalent - Notes	
	No equivalent command at present.

8753 Command	Description	Range	Query Response
DELR<num>	Makes the indicated marker the delta reference.	Integers 15	<0|1>><LF
DELRFIXM	Makes the fixed marker the delta reference.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
Step 1 CALC:MARK:REF	If the reference marker is not turned ON, use this command to set the reference marker to ON.
Step 2 CALC:MARK:REF:X	Set the reference marker to the correct position.
Step 3 CALC:MARK:DELT	Turn marker delta mode ON for the marker that will report the delta value measured from the reference marker.

PNA COM Equivalent - Notes

No equivalent command at present.

DELA

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>DELA</td><td>Displays the data formatted as group delay.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

CALC:FORM MLIN	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes

Format_Property	Sets (or returns) the display format of the measurement.

DISP

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>DISPDATA</td><td>Data only.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>DISPDATM</td><td>Data and memory.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>DISPDDM</td><td>Data divided by memory (linear division, log subtraction). See also "DIVI."</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>DISPDMM</td><td>Data minus memory (linear subtraction). See also "MINU."</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>DISPMEMO</td><td>Memory only.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

CALC:MATH:FUNC	Read-Write math operations on the currently selected measurement and the trace stored in memory.

PNA COM Equivalent - Notes

Trace_Math_Property	Performs math operations on the measurement object and the trace stored in memory.

DIVI

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>DIVI</td><td>Data divided by memory (linear division, log subtraction). See also "DISPDDM."</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

CALC:MATH:FUNC	Read-Write math operations on the currently selected measurement and the trace stored in memory.

PNA COM Equivalent - Notes

Trace_Math_Property	Performs math operations on the measurement object and the trace stored in memory.

DONE

8753 Command	Description	Range	Query Response
DONE	Done with a class of standards, during a calibration. Only needed when multiple standards are measured to complete the class.	N/A	N/A

Notes

Since this action is performed automatically in PNA, this command is no longer necessary.

ECALAB?

8753 Command	Description	Range	Query Response
ECALAB	Queries the analyzer for the currently selected module	N/A	<0|1><LF

PNA SCPI Equivalent - Notes

No equivalent command at this time

PNA COM Equivalent - Notes

IsECALModuleFound Property — Tests communication between the PNA and the specified ECal module.

ECALDONE

8753 Command	Description	Range	Query Response
ECALDONE	Designed to be used in a polling loop to determine if the ECAL operation is finished.	N/A	N/A

PNA SCPI Equivalent - Notes

1. SENS:CORR:COLL:ACQ ECAL<A|B> — Measures the ECAL A module
2. *OPC? — Operation Complete query

PNA COM Equivalent - Notes

COM methods do not return until the cal is complete

ECALFUL2

8753 Command	Description	Range	Query Response
ECALFUL2	Performs an full 2-port ECAL	N/A	<0|1><LF>

PNA SCPI Equivalent - Notes

1. SENS:CORR:COLL:METH SPARSOLT — Sets the calibration method to SOLT
2. SENS:CORR:COLL:ACQ ECAL<A|B> — Measures the ECAL module

PNA COM Equivalent - Notes

DoECAL2Port Method — Does a 2-Port calibration using an ECAL module.

ECALISOAVG

8753 Command	Description	Range	Query Response
ECALISOAVG	Sets the number of averages in the	1-999	<num><LF>

	ECAL isolation averages function		

PNA SCPI Equivalent - Notes

SENS:AVER:COUN	Sets the number of measurement sweeps to combine for an average.

PNA COM Equivalent - Notes

Averaging Factor Property	Specifies the number of measurement sweeps to combine for an average

---

ECALMODINF

8753 Command	Description	Range	Query Response
ECALMODINF	Returns string information on the selected ECAL module.	N/A	<array><LF>

PNA SCPI Equivalent - Notes

	No equivalent command at this time

PNA COM Equivalent - Notes

Get ECAL Module Info Method	Returns the following information about the connected ECAL module: model number, serial number, connector type, calibration date, min and max frequency.

ECALOMII

8753 Command	Description	Range	Query Response
ECALOMII	Set omit isolation ON or OFF	N/A	<0|1><LF

PNA SCPI Equivalent - Notes

SENS:CORR:ISOL	Turns isolation cal ON or OFF during Full 2-port (or ECAL) calibration.

PNA COM Equivalent - Notes

ECALIsolation Property	Specifies whether the acquisition of the ECal calibration should include isolation or not.

ECALS11

8753 Command	Description	Range	Query Response
ECALS11	Performs a S11 ECAL	N/A	N/A

PNA SCPI Equivalent - Notes

1. SENS:CORR:COLL:METH REFL3	Sets the calibration method to 1-port
2. SENS:CORR:COLL:ACQ ECAL<A|B>	Measures the ECAL module

PNA COM Equivalent - Notes

DoECAL1Port Method	Does a 1-Port calibration using an ECAL module.

ECALS22

8753 Command	Description	Range	Query Response
ECALS22	Performs a S22 ECAL	N/A	N/A

PNA SCPI Equivalent - Notes

1. SENS:CORR:COLL:METH REFL3	Sets the calibration method to 1-port
2. SENS:CORR:COLL:ACQ ECAL<A|B>	Measures the ECAL module

PNA COM Equivalent - Notes			
DoECAL1Port Method		Does a 1-Port calibration using an ECAL module.	

EDIT

8753 Command	Description	Range	Query Response
EDITDONE	Done editing list frequency, limit table, cal sensor table, or power loss list.	N/A	N/A
EDITLIML	Begins editing limit table.	N/A	N/A
EDITLIST	Begins editing list frequency table.	N/A	N/A
Notes			
		Since these actions are performed automatically in PNA when working with a limit table, these commands are no longer necessary.	

ELED

8753 Command	Description	Range	Query Response
ELED<num>[S]	Sets the electrical delay offset.	±10 seconds	<num><LF
PNA SCPI Equivalent - Notes			
CALC1:CORR:EDEL:TIME		Read-Write the electrical delay for the selected measurement.	
PNA COM Equivalent - Notes			
Electrical_Delay_Property		Sets the Electrical Delay	

ESE

8753 Command	Description	Range	Query Response
ESE<num>	Enables the selected event-status register bits to be summarized by bit 5 in the status byte. An event-status register bit is enabled when the corresponding bit in the operand <num> is set.	integers 0255	<num><LF
PNA SCPI Equivalent - Notes			
*ESE		Sets bits in the standard event status enable register. Replace "ESE" with "*ESE".	
PNA COM Equivalent - Notes			
		No equivalent command at present.	

ESR?

8753 Command	Description	Range	Query Response
ESR?	Query only. Outputs event-status register.	N/A	<num><LF

PNA SCPI Equivalent - Notes

*ESR	Returns the results of the standard event enable register. The register is cleared after reading it. Replace "ESR?" with "*ESR?".

PNA COM Equivalent - Notes

	No equivalent command at present.

EXTM

8753 Command	Description	Range	Query Response
EXTMDATA	Adds error corrected data (real and imaginary pairs) along with the other files.	N/A	<0|1>><LF
EXTMDATO	Selected data arrays only (real and imaginary pairs), without instrument states or calibrations. Always saves the data array, even if it hasnt been selected.	N/A	<0|1>><LF
EXTMFORM	Formatted trace data. Uses currently selected format for data.	N/A	<0|1>><LF
EXTMRAW	Raw data arrays (real and imaginary pairs).	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:DATA:CUST	Read-Write either measurement data or memory data.

PNA COM Equivalent - Notes

Get_Data_Method	Retrieves data.

8753 Command	Description	Range	Query Response
EXTMGRAP	User graphics.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

	No equivalent command at present.

PNA COM Equivalent - Notes

PrintToFile_Method	Saves the screen data to bitmap (.bmp) file of the screen.

EXTT

8753 Command	Description	Range	Query Response
EXTT	Activates or deactivates the external trigger mode. OPC-compatible.	N/A	<0|1>><LF
EXTTPOIN	Sets the external trigger to auto-trigger on point. OPC-compatible.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

TRIG:SOUR	Read-Write the source of the sweep trigger signal.
SENS:SWE:TRIG:POIN	Read-Write whether the specified channel will measure one point when triggered or all of the measurements in the channel.

PNA COM Equivalent - Notes

Trigger_Signal_Property	Sets or returns the trigger source.
Trigger_Mode_Property	Determines the measurement that occurs when a trigger signal is sent to the channel.

8753 Command	Description	Range	Query Response
EXTTHIGH	Sets the external trigger line high.	N/A	N/A
EXTTLOW	Sets the external trigger line low.	N/A	N/A

PNA SCPI Equivalent - Notes

No equivalent command at present.

PNA COM Equivalent - Notes

No equivalent command at present.

FORM

8753 Command	Description	Range	Query Response
FORM1	The analyzer's internal binary format, 6 bytes-per-data point. The array is preceded by a four-byte header. The first two bytes represent the string "#A", the standard block header. The second two bytes are an integer representing the number of bytes in the block to follow. FORM1 is best applied when rapid data transfers, not to be modified by the computer nor interpreted by the user, are required.	N/A	N/A
FORM2	IEEE 32-bit floating-point format, 4 bytes-per-number, 8 bytes-per-data point. The data is preceded by the same header as	N/A	N/A

	in FORM1. Each number consists of a 1-bit sign, an 8-bit biased exponent, and a 23-bit mantissa. FORM2 is the format of choice if your computer is not a PC, but supports single-precision floating-point numbers.		
FORM3	IEEE 64-bit floating-point format, 8 bytes-per-number, 16 bytes-per-data point. The data is preceded by the same header as in FORM1. Each number consists of a 1-bit sign, an 11-bit biased exponent, and a 52-bit mantissa. This format may be used with double-precision floating-point numbers. No additional precision is available in the analyzer data, but FORM3 may be a convenient form for transferring data to your computer.	N/A	N/A
FORM4	ASCII floating-point format. The data is transmitted as ASCII numbers. There is no header. The analyzer always uses FORM4 to transfer data that is not related to array transfers (i.e. marker responses and instrument settings). Data is comma delimited.	N/A	N/A
FORM5	PC-DOS 32-bit floating-point format with 4 bytes-per-number, 8 bytes-per-data point. The data is preceded by the same header as in FORM1. The byte order is reversed with respect to FORM2 to	N/A	N/A

	comply with PC-DOS formats. If you are using a PC-based controller, FORM5 is the most effective format to use.		

PNA SCPI Equivalent - Notes

	FORM1, the 8753 analyzer's internal binary format, is not compatible with PNA.
format:data	For FORM2, FORM3, and FORM4, use this SCPI command.
Step 1 format:data	For FORM5, format the data to 32-bit with the SCPI command in Step 1. Then, swap the bits with the SCPI command in Step 2.
Step 2 format:border	

PNA COM Equivalent - Notes

	No equivalent command at present.

FRER

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>FRER</td><td>Places the analyzer in GPIB free run mode. (Same as continuous sweep trigger mode.) See "CONT."</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

initiate:continuous	Read-Write the sweep triggering mode.

PNA COM Equivalent - Notes

Continuous_Method	Read-Write the sweep triggering mode.

FWD

8753 Command	Description	Range	Query Response
FWDI	Selects the forward isolation calibration class during a 2-port calibration sequence.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:CORR:COLL	Measure the specified standard from the selected calibration kit. See "STAN5".

PNA COM Equivalent - Notes

AcquireCalStandard2_Method	Measure the specified standard from the selected calibration kit.

8753 Command	Description	Range	Query Response
FWDM	Selects the forward match calibration class during a 2-port calibration sequence.	N/A	N/A
FWDT	Selects the forward transmission calibration class during a 2-port	N/A	N/A

	calibration sequence.	
Notes		Both the forward match and the forward transmission are measured automatically during a 2-port calibration on the PNA Series Network Analyzers.

HOLD

8753 Command	Description	Range	Query Response
HOLD	Puts the sweep trigger into hold mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
initiate:continuous	Read-Write the sweep triggering mode. Set the sweep trigger mode to "OFF".

PNA COM Equivalent - Notes	
Hold_Method	Put the sweep trigger into hold mode.

IDN?

8753 Command	Description	Range	Query Response
IDN?	Query only. Outputs the identification string: AGILENT TECHNOLOGIES ,87NNEX,xxxxxxxxxx ,X.XX where 87NNEX is the model number of the instrument, xxxxxxxxxx is the serial number of the instrument, and X.XX is the firmware revision of the instrument.	N/A	See command description.

PNA SCPI Equivalent - Notes	
*IDN?	Returns a string that uniquely identifies the analyzer. Replace "IDN?" with "*IDN?".

PNA COM Equivalent - Notes	
Application_Property	Returns the name of the Analyzer making measurements on the channel.

IF

8753 Command	Description	Range	Query Response
IFBIHIGH IFBILOW	Tests the specified GPIO bit. If HIGH / LOW invokes the sequence which follows.	N/A	N/A

PNA SCPI Equivalent - Notes	
CONT:AUX:PASS:LOG	Sets the logic of the PassFail line (pin 12) on

the AUX IO connector. This line is connected internally to the PassFail line of the Material Handler IO (pin 33).

PNA COM Equivalent - Notes

PassFailLogic Property	Sets the logic of the PassFail line (pin 12) on the AUX IO connector. This line is connected internally to the PassFail line of the Material Handler IO (pin 33).

IFBW

8753 Command	Description	Range	Query Response
IFBW<num>[HZ]	Sets the IF bandwidth.	Choose from 10, 30, 100, 300, 1000, 3000, 3700, 6000	<num><LF

PNA SCPI Equivalent - Notes

SENS:BWID	Read-Write the bandwidth of the digital IF filter to be used in the measurement.

PNA COM Equivalent - Notes

IF_Bandwidth_Property	Sets or returns the IF Bandwidth of all measurements in a channel.

IMAG

8753 Command	Description	Range	Query Response
IMAG	Selects the imaginary display format.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:FORM	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes

Format_Property	Sets (or returns) the display format of the measurement.

INPU

- INPUCALC
- INPUCALK
- INPUDATA
- INPUFORM
- INPULEAS
- INPUPMCAL1
- INPUPMCAL2
- INPURAW1
- INPURAW2
- INPURAW3
- INPURAW4

INPUCALC

8753 Command	Description	Range	Query Response
INPUCALC<num><array>	Inputs an error coefficient array <num>	N/A	N/A

PNA SCPI Equivalent - Notes

CALC:DATA:	Writes Measurement data, Memory data, or Error terms

PNA COM Equivalent - Notes

Put_Error_Term_Method	Puts variant error term data into the error-correction buffer.
Put_Error_Term_Complex_Method	Puts typed error term data into the error-correction buffer

8753 Command	Description	Range	Query Response
INPUDATA	Inputs an error corrected data array, using the current setting of the FORM command.	N/A	N/A
INPUFORM	Inputs a formatted data array, using the current setting of the FORM command.	N/A	N/A
INPURAW1	Inputs raw data array 1 (S11 data). After the data is received, the analyzer stops sweeping, error-corrects the data, then formats and displays the data.	N/A	N/A
INPURAW2	Inputs raw data array 2 (S21 data). After the data is received, the analyzer stops sweeping, error-corrects the data, then formats and displays the data.	N/A	N/A
INPURAW3	Inputs raw data array 3 (S12 data). After the data is received, the analyzer stops sweeping, error-corrects the data, then formats and displays the data.	N/A	N/A
INPURAW4	Inputs raw data array 4 (S22 data). After the data is received, the analyzer stops sweeping, error-corrects the data, then formats and displays the data.	N/A	N/A

PNA SCPI Equivalent - Notes

Step 1 CALC:DATA:	Input the data array.
Step 2 SENS:CORR	If the downloaded data array is error corrected, then error corrections need to be turned OFF. If not, an additional set of corrections will be applied to the downloaded data.

PNA COM Equivalent - Notes	
Step 1aPut_Data_Complex_Method	Input raw data.
Step 1bPut_Data_Complex_Method	Input formatted data
Step 2 Put_Data_Complex_Method	If the downloaded data array is error corrected, then error corrections need to be turned OFF. If not, an additional set of corrections will be applied to the downloaded data.

INPUPMCAL

8753 Command	Description	Range	Query Response
INPUPMCAL<array>	Inputs an power meter calibration arrays for channels 1 and 2 in FORM4 only	N/A	N/A

PNA SCPI Equivalent - Notes	
SOUR1:POW:CORR:DATA	Writes and reads source power calibration data

PNA COM Equivalent - Notes	
putSourcePowerCalData Method	Inputs source power calibration data (as variant data type) to this channel for a specific source port.
putSourcePowerCalDataScalar	Inputs source power calibration data (as scalar values) to this channel for a specific source port.

The following commands are not currently available.

8753 Command	Description	Range	Query Response
INPUCALK<array>	Inputs a cal kit array in FORM1 only. Can be read out with the OUTCALK command. After the transfer, the data should be saved into the user cal kit area with the SAVEUSEK command.	N/A	N/A
INPULEAS<learnstring>	Inputs a learn string in FORM1 only. Can be read out with the OUTPLEAS command, or with INPULEAS?.	N/A	<data><LF >

LIM

8753 Command	Description	Range	Query Response
LIMS	Sets the limit stimulus break point.	Stimulus range.	Currently this command can be queried by sending the command by the OUTPACTI command.

PNA SCPI Equivalent - Notes	
CALC:LIM:SEGM1:STIM:STAR	Read-Write the start (beginning) of the X-axis stimulus value.
CALC:LIM:SEGM1:AMPL:STOP	Read-Write the stop (end) of the X-axis stimulus value.

PNA COM Equivalent - Notes	
Begin_Stimulus_Property	Specifies the beginning X-axis value of the Limit Line.
End_Stimulus_Value	Specifies the end X-axis value of the Limit Line.

8753 Command	Description	Range	Query Response
LIMD	Sets the limit delta value while editing a limit line segment.	Amplitude range.	Currently this command can be queried by sending the command by the OUTPACTI command.
LIML	Sets the lower limit value.	Amplitude range.	Same as above.
LIMM	Sets the middle limit value.	Amplitude range.	Same as above.
LIMU	Sets the upper limit value.	Amplitude range.	Same as above.

PNA SCPI Equivalent - Notes	
CALC:LIM:DATA	Read-Write data for limit lines.

PNA COM Equivalent - Notes	
LimitSegment_Object	Make a limit line object.

LIMI

- LIMIAMPO
- LIMILINE
- LIMIMAOF
- LIMISTIO
- LIMITEST

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>LIMILINE<ON|OFF></td><td>Turns the display of the limit lines on and off.</td><td>N/A</td><td><0|1><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes	
CALC:LIM:DISP:STAT	Read-Write the display of limit lines ON or OFF.

PNA COM Equivalent - Notes	
LineDisplay_Property	Turns the display of limit lines ON or OFF.

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>LIMITEST<ON|OFF></td><td>Turns limit testing on and off.</td><td>N/A</td><td><0|1><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes	
CALC:LIM:STAT	Read-Write limit line **testing** ON or OFF.

PNA COM Equivalent - Notes

State_Property	Turns an object ON and OFF.

8753 Command	Description	Range	Query Response
LIMIAMPO<num>[HZ|DB]	Enters the limit line amplitude offset.	Amplitude range.	<num>><LF
LIMIMAOF	Marker to limit offset. Centers the limit lines about the current marker position using the limit amplitude offset function.	N/A	N/A
LIMISTIO<num>[HZ|DB]	Enters the stimulus offset of the limit lines.	Stimulus range.	<num>><LF

Notes

These commands currently are not available.

LIMT

8753 Command	Description	Range	Query Response
LIMTFL	Makes the segment a flat line.	N/A	<0|1>><LF
LIMTSL	Makes the segment a sloping line.	N/A	<0|1>><LF
LIMTSP	Makes the segment a single point.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:LIM:DATA	Read-Write data for limit lines.

PNA COM Equivalent - Notes

LimitSegment_Object	Make a limit line object.

LINFREQ

8753 Command	Description	Range	Query Response
LINFREQ	Selects a linear frequency sweep.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:SWE:TYPE	Read-Write the type of analyzer sweep mode.

PNA COM Equivalent - Notes

Sweep_Type_Property	Sets the type of X-axis sweep that is performed on a channel.

LINM

8753 Command	Description	Range	Query Response
LINM	Selects the linear magnitude display format.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:FORM	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes

Format_Property	Sets (or returns) the display format of the measurement.

LIS

- LISFREQ
- LISIFBWM
- LISPWRM

8753 Command	Description	Range	Query Response
LISFREQ	Selects the list frequency sweep mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:SWE:TYPE	Selects the sweep type.
PNA COM Equivalent - Notes	
Sweep_Type_Property	Selects the sweep type.

8753 Command	Description	Range	Query Response
LISIFBWM<ON|OFF>	Enables/disables the IFBW setting for a list-frequency table in swept list mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:SEGM:BWID:CONT	Read-Write whether the IF Bandwidth resolution can be set independently for each segment.
SENS:SEGM:BWID	Read-Write the IFBandwidth for the specified segment.
PNA COM Equivalent - Notes	
IF_Bandwidth_Option_Property	Enables the IFBandwidth to be set on individual sweep segments.
IF_Bandwidth_Property	Sets or returns the IF Bandwidth of the segment.

8753 Command	Description	Range	Query Response
LISPWRM<ON|OFF>	Enables/disables the power setting for a list-frequency table in swept list mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:SEGM:POW:CONT	Read-Write whether Power Level can be set independently for each segment.
SENS:SEGM:POW	Read-Write the Port Power level for the specified segment.
PNA COM Equivalent - Notes	
Source_Power_Option_Property	Enables the source power to be set on individual sweep segments.
Test_Port_Power_Property	Sets or returns the RF power level of the segment.

LISTTYPE

8753 Command	Description	Range	Query Response
LISTTYPELSTP	Selects the stepped list mode for use with a list-frequency table.	N/A	<0|1><LF>
LISTTYPELSWP	Selects the swept list mode for use with a list-frequency table.	N/A	<0|1><LF>

PNA SCPI Equivalent - Notes	
SENS:SWE:GEN	Read-Write sweep as Stepped or Analog.
PNA COM Equivalent - Notes	
Sweep_Generation_Mode_Property	Sets the method used to generate a sweep: continuous ramp (analog) or discrete steps (stepped).

LOAD

8753 Command	Description	Range	Query Response
LOAD<num>	Loads the file from disk using the file name provided by the preceding TITF<num>; command. The actual file loaded depends on the file title in the file position specified by the TITF<num> command. Requires pass control mode when using the GPIB port.	integers 15	N/A

PNA SCPI Equivalent - Notes	
MMEM:LOAD	Write-only to load the specified file.
PNA COM Equivalent - Notes	
Recall_Method	Recalls a measurement state, calibration state, or both.

LOGM

8753 Command	Description	Range	Query Response
LOGM	Selects the log magnitude display format.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:FORM	Read-Write the display format for the measurement.
PNA COM Equivalent - Notes	
Format_Property	Sets (or returns) the display format of the measurement.

MANTRIG

8753 Command	Description	Range	Query Response

<table>
<tr><td>MANTRIG</td><td>Sets the trigger mode to manual trigger on point. OPC-compatible.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

Step 1 TRIG:SOUR	Set the trigger source to manual.
Step 2 SENS:SWE:TRIG:POIN	Set the trigger mode to point.

PNA COM Equivalent - Notes

Step 1 Trigger_Signal_Property	Set the trigger source to manual.
Step 2 Trigger_Mode_Property	Set the trigger mode to point.

MARK

- MARK<1|2|3|4|5>
- MARKBUCK
- MARKCENT
- MARKCONT
- MARKCOUP
- MARKCW
- MARKDELA
- MARKDISC
- MARKFAUV
- MARKFSTI
- MARKFVAL
- MARKMAXI
- MARKMIDD
- MARKMINI
- MARKOFF
- MARKREF
- MARKSPAN
- MARKSTAR
- MARKSTIM
- MARKSTOP
- MARKUNCO
- MARKZERO

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>MARK<1|2|3|4|5><num></td><td>Makes the selected marker active and sets its stimulus value.</td><td>Stimulus range. For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ±1/frequency step. For CW time sweep, transform on: ±1/time step.</td><td><num><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

Step 1 CALC:MARK	Set the specified marker ON. Note: CALCulate commands act on the selected measurement. You must have a measurement defined and selected before a marker can be turned on. To define a measurement use CALCulate<cnum>:PARameter:DEFine <Mname>,<param>. Select the measurement for each channel using CALCulate<cnum>:PARameter:SELect <Mname>.
Step 2 CALC:MARK:X	Set the marker's X-axis value (frequency,

power, or time).

PNA COM Equivalent - Notes

Stimulus_Property — Sets and reads the X-Axis value of the marker.

8753 Command	Description	Range	Query Response
MARKBUCK<num>	Places the active marker on a specific sweep point (bucket). <num> is the bucket number.	0 to (number-of-points - 1). For example, on a 201 point sweep, <num> can range from 0 to 200.	<num><LF

PNA SCPI Equivalent - Notes

No equivalent command at present.

PNA COM Equivalent - Notes

Bucket_Number_Property — Sets or returns the bucket number (data point) for the active marker.

8753 Command	Description	Range	Query Response
MARKCENT	Sets the **center** stimulus value to that of the active marker's stimulus value.	N/A	N/A
MARKSTAR	Sets the **start** stimulus to that of the active marker's.	N/A	N/A
MARKSTOP	Sets the **stop** stimulus to that of the active marker's.	N/A	N/A
MARKREF	Sets the **reference value** to that of the active marker's amplitude.	N/A	N/A

PNA SCPI Equivalent - Notes

CALC:MARK:SET — Read-Write the selected instrument setting to assume the value of the specified marker.

PNA COM Equivalent - Notes

Set_Center_Method — Changes the analyzer's **center** frequency to the X-axis position of the marker. The start frequency stays the same and the stop frequency adjusts.

Set_Start_Method — Changes the analyzer's **start** frequency to the X-axis position of the marker. The stop frequency stays the same and the frequency span adjusts.

Set_Stop_Method — Changes the analyzer's **stop** frequency to the X-axis position of the marker. The start frequency stays the same and the frequency span adjusts.

SetReferenceLevel_Method — Changes the measurement's **reference level** to the marker's Y-axis value.

8753 Command	Description	Range	Query Response

<table>
<tr><td>MARKDELA</td><td>Sets electrical length so group delay is zero at the active marker’s stimulus.</td><td>N/A</td><td>N/A</td></tr>
</table>

PNA SCPI Equivalent - Notes

No equivalent command at present.

PNA COM Equivalent - Notes

SetElectricalDelay_Method	Changes the measurement’s electrical delay value to the marker’s delay value.

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>MARKMAXI</td><td>Search for trace maximum on the current channel. Same as SEAMAX.</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>MARKMINI</td><td>Search for trace minimum on the current channel. Same as SEAMIN.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

Step 1 CALC:MARK	If a marker is not currently turned ON, set marker to ON with this command.
Step 2 CALC:MARK:FUNC:EXEC	Write-only to immediately execute (perform) the specified search function.

PNA COM Equivalent - Notes

Search_Max_Method	Searches the marker domain for the **maximum** value.
Search_Min_Method	Searches the marker domain for the **minimum** value.

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>MARKCONT</td><td>Places the markers continuously on the trace, not on discrete points (interpolates the marker values between discrete points).</td><td>N/A</td><td><0|1>><LF</td></tr>
<tr><td>MARKDISC</td><td>Places the markers on the discrete measurement points.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

CALC:MARK:DISC	Read-Write the specified marker as either interpolate data or not.

PNA COM Equivalent - Notes

Interpolate_Markers_Method	Turns **All** Marker Interpolation ON and OFF for the measurement.

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>MARKOFF</td><td>Turns all markers and marker functions off.</td><td>N/A</td><td><0|1>><LF</td></tr>
</table>

PNA SCPI Equivalent - Notes

CALC:MARK:AOFF	Write-only all markers off for selected measurement.

PNA COM Equivalent - Notes

DeleteAllMarkers_Method	Turn markers OFF by deleting all of the markers from the measurement.

8753 Command	Description	Range	Query Response
MARKCW	Sets the CW frequency to the active marker's frequency.	N/A	N/A

PNA SCPI Equivalent - Notes

No equivalent command at present.

PNA COM Equivalent - Notes

Set_CW_Method	Changes the analyzer to sweep type CW mode and makes the CW frequency the marker's frequency.

8753 Command	Description	Range	Query Response
MARKFSTI<num>	Sets the stimulus position of the fixed marker.	Stimulus range. For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ±1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes

Step 1 CALC:MARK	If the marker is not turned ON, set the marker ON.
Step 2 CALC:MARK:TYPE	Set the marker type to "Fixed".
Step 3 CALC:MARK:X	Set the position of the marker.

PNA COM Equivalent - Notes

Step 1 Type_Marker_Property	Sets and reads the marker type.
Step 2 Stimulus_Property	Sets and reads the X-Axis value of the marker.

8753 Command	Description	Range	Query Response
MARKMIDD	Makes the marker amplitude the limit segment middle value during a limit segment edit.	N/A	N/A
MARKSPAN	Sets the span for the entire trace to that of the span between the active marker and the	N/A	N/A

	delta reference marker.		
MARKSTIM	During a limit segment edit, sets the limit stimulus break point to that of the active marker's.	N/A	N/A
MARKZERO	Places the fixed marker at the active marker position and makes it the delta reference.	N/A	N/A

Notes

These functions require multiple SCPI or COM commands along with appropriate math calculations.

8753 Command	**Description**	**Range**	**Query Response**
MARKCOUP	Couples the markers between the channels, as opposed to MARKUNCO.	N/A	<0|1>><LF
MARKUNCO	Uncouples the markers between channels, as opposed to MARKCOUP.	N/A	<0|1>><LF
MARKFAUV	Sets the auxiliary value of the fixed marker position. Works in coordination with MARKFVAL and MARKFSTI.	Amplitude range. Same as MARKFVAL.	<num><LF
MARKFVAL	Sets the value of the fixed marker position.	Amplitude range. For log mag: ± 500 dB. For phase: ± 500 degrees. For Smith chart and Polar: ± 500 units. For linear magnitude: ± 500 units. For SWR: ± 500 units. The scale is always positive, and has minimum values of 0.001dB, 10e-12 degrees, 10e-15 seconds, and 10 picounits.	<num><LF

Notes

These commands are currently not supported.

MEAS

8753 Command	Description	Range	Query Response
MEASA	Measures and displays input A on the active channel.	N/A	<0|1>><LF
MEASB	Measures and displays input B on the active channel.	N/A	<0|1>><LF
MEASR	Measures and displays input R on the active channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
Step 1 CALC:PAR:DEF	Create the measurement.
Step 2 DISP:WIND	If a new window will be used to display the measurement, then create a window.
Step 3 DISP:WIND:TRAC:FEED	Display the measurement in the window.

PNA COM Equivalent - Notes	
CreateMeasurement_Method	Create and display the measurement.

MEASTAT

8753 Command	Description	Range	Query Response
MEASTAT<ON|OFF>	Turns trace statistics on and off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
	No equivalent command at present.

PNA COM Equivalent - Notes	
Show_Statistics_Property	Displays and hides the measurement statistics (peak-to-peak, mean, standard deviation) on the screen.

MINMAX

8753 Command	Description	Range	Query Response
MINMAX<ON|OFF>	Enables/disables min/max recording per segment. Min and max values are recorded per limit segment. Limit testing need not be active.	N/A	<0|1>><LF

Notes	
	This command is not available on PNA.

MINU

8753 Command	Description	Range	Query Response
MINU	Data minus memory (linear subtraction). See also "DISPDMM."	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:MATH:FUNC	Read-Write math operations on the currently selected measurement and the trace stored in memory.

PNA COM Equivalent - Notes

Trace_Math_Property	Performs math operations on the measurement object and the trace stored in memory.

NUMG

8753 Command	Description	Range	Query Response
NUMG<num>	Activates the indicated number of groups of sweeps. A group is whatever is needed to update the current parameter once. This function restarts averaging if it is enabled. OPC-compatible.	Integers 1999.	N/A

PNA SCPI Equivalent - Notes	
Step 1 SENS:SWE:GRO:COUN	Set the number of groups.
Step 2 SENS:SWE:MODE	Set the trigger mode to groups.

PNA COM Equivalent - Notes	
Number Of Groups Method	Sets the Number of trigger signals the channel will receive. After the channels has received that number of trigger signals, the channel switches to Hold mode.

NUMR

8753 Command	Description	Range	Query Response
NUMR	Sets the number of power meter readings per point in a power calibration	integers 1 to 100	<num><LF

PNA SCPI Equivalent - Notes	
SOUR:POW:CORR:COLL:AVER	Specifies how many power readings are taken at each frequency point (averaging factor) during a source power cal acquisition sweep.

PNA COM Equivalent - Notes	
ReadingsPerPoint Property	For purpose of averaging during source power cal, specifies how many power readings are taken at each frequency point (Averaging factor).

OMII

8753 Command	Description	Range	Query Response
OMII	Omits the isolation step of a calibration sequence.	N/A	N/A

PNA SCPI Equivalent - Notes	
SENS:CORR:ISOL	Read-Write isolation cal ON or OFF during Full 2-port calibration.

PNA COM Equivalent - Notes	
Acquire Cal Standard2 Method	To omit Isolation from a 2-port calibration, do

		not Acquire a cal standard for naSOLT_Isolation	

OPC

8753 Command	Description	Range	Query Response
OPC	Operation complete. Reports the completion of the next command received by setting bit 0 in the event-status register, or by replying to an interrogation if OPC? is issued.	N/A	<0|1>><LF>

PNA SCPI Equivalent - Notes	
*OPC	Operation complete command. Replace "OPC" with "*OPC".
*OPC?	Operation complete query. Replace "OPC?" with "*OPC?".

PNA COM Equivalent - Notes	
	No equivalent command at present.

OUTP

OUTPACTI	OUTPFAIP	OUTPMARK	OUTPRAF
OUTPAMAX	OUTPFORE	OUTPMEMF	OUTPRAW
OUTPAMIN	OUTPFORM	OUTPMEMO	OUTPRFFR
OUTPAPER	OUTPICAL	OUTPMSTA	OUTPSEGAF
OUTPCALC	OUTPIDEN	OUTPMWID	OUTPSEGAM
OUTPCALK	OUTPIPMCL	OUTPMWIL	OUTPSEGF
OUTPCHAN	OUTPKEY	OUTPOPTS	OUTPSEGM
OUTPDATA	OUTPLEAS	OUTPPLOT	OUTPSEQ
OUTPDATF	OUTPLIM	OUTPPMCAL	OUTPSERN
OUTPDATP	OUTPLIMF	OUTPPRE	OUTPSTAT
OUTPDATR	OUTPLIML	OUTPPRIN	OUTPTITL
OUTPERRO	OUTPLIMM	OUTPPRNALL	

8753 Command	Description	Range	Query Response
OUTPAPER	Outputs the smoothing aperture in stimulus units, rather than as a percentage.	N/A	<num><LF

PNA SCPI Equivalent - Notes	
Step 1 CALC:SMO:APER	Output the smoothing aperture.
Step 2 SENS:FREQ:SPAN	Output the span.
Step 3	Multiply the smoothing aperture as a decimal number with the span to produce the smoothing aperture in stimulus units.

PNA COM Equivalent - Notes	
Step 1 Smoothing Aperture Property	Output the smoothing aperture.
Step 2 Frequency Span Property	Output the span.
Step 3	Multiply the smoothing aperture as a decimal

number with the span to produce the smoothing aperture in stimulus units.

8753 Command	Description	Range	Query Response
OUTPCALC	Outputs the selected error coefficient array for the active cal on the active channel.	Two-digit integers 0112	<array><LF

PNA SCPI Equivalent - Notes

CALC:DATA — Output the error coefficient array.

PNA COM Equivalent - Notes

GetErrorTerm Method — Retrieves error term data for the active calibration.

GetErrorTerm Complex Method — Retrieves error term data in complex pairs from the error correction buffer.

8753 Command	Description	Range	Query Response
OUTPCALK	Outputs the currently active calibration kit, as a string of less than 1000 bytes. The data is in FORM1.	N/A	<$><LF >

PNA SCPI Equivalent - Notes

SENS:CORR:COLL:CKIT:NAME — Read-Write a name for the selected calibration kit.

PNA COM Equivalent - Notes

Name CalKit Property — Sets and Returns a name for the selected calibration kit.

8753 Command	Description	Range	Query Response
OUTPDATA	Outputs the error-corrected data from the active channel in real/imaginary pairs.	N/A	<array><LF

PNA SCPI Equivalent - Notes

CALC:DATA — Output the error-corrected data array.

PNA COM Equivalent - Notes

GetNAComplex Method — Output the error-corrected data array.

8753 Command	Description	Range	Query Response
OUTPFORM	Outputs the formatted display data array from the active channel, in current display units.	N/A	<array><LF

PNA SCPI Equivalent - Notes

CALC:DATA — Output the data array.

PNA COM Equivalent - Notes

GetData Method — Output the data array.

8753 Command	Description	Range	Query Response
OUTPIDEN	Outputs the identification string	N/A	<$><LF >

for the analyzer in the form: AGILENT TECHNOLOGIES ,87NNEX,xxxxxxxxxx ,X.XX where 87NNEX is the model number of the instrument, xxxxxxxxxx is the serial number of the instrument, and X.XX is the firmware revision of the instrument. (Same as the "IDN?" command.)

PNA SCPI Equivalent - Notes

*IDN?	Returns a string that uniquely identifies the analyzer.

PNA COM Equivalent - Notes

Application Property	Returns the name of the Analyzer making measurements on the channel.

8753 Command	Description	Range	Query Response
OUTPIPMCL	Outputs the interpolated power meter calibration array for channel 1 or channel 2. Values are returned as 100 times the interpolated power meter reading in dB. This is an ASCII transfer (FORM4).	Integers 1 or 2.	<array><LF

PNA SCPI Equivalent - Notes

SOUR:POW:CORR:DATA	Writes and reads source power calibration data.

PNA COM Equivalent - Notes

get SourcePowerCalData Method get SourcePowerCalDataScalar Method	Retrieves requested source power calibration data, if it exists, from this channel.

8753 Command	Description	Range	Query Response
OUTPLIM	Outputs the status of the limit test for the channel selected with <num>.	Integers 14	<0|1|-1><LF

PNA SCPI Equivalent - Notes

STAT:QUES:LIM1:COND?	Check status bit to determine status of the limit test.

PNA COM Equivalent - Notes

No equivalent command at present.

8753 Command	Description	Range	Query Response
OUTPLIML	Outputs the limit test results for each point in the sweep. This is an ASCII transfer.	N/A	<array><LF
OUTPLIMM	Outputs the limit test results at the active marker.	N/A	<num,num,num,num><LF

PNA SCPI Equivalent - Notes

	No equivalent command at present.

PNA COM Equivalent - Notes

Get Test Result Method	Returns the result of limit line testing.

8753 Command	Description	Range	Query Response
OUTPMARK	Outputs the active marker values. The first two numbers are the marker response values, and the last is the stimulus value.	N/A	<num,num,num><LF>

PNA SCPI Equivalent - Notes

CALC:MARK:X	Read-Write the marker's X-axis value (frequency, power, or time).
CALC:MARK:Y?	Read-only the marker's Y-axis value.

PNA COM Equivalent - Notes

Stimulus Property	Sets and reads the X-Axis value of the marker.
Value Property	Reads the Y-Axis value of the marker.

8753 Command	Description	Range	Query Response
OUTPMEMO	Outputs the memory trace from the active channel. The data is in real/imaginary pairs, and can be treated the same as data read with the OUTPDATA command.	N/A	<array><LF

PNA SCPI Equivalent - Notes

CALC:DATA	Read-Write either measurement data or memory data. When querying memory, you must first store a trace into memory using CALCuate<cnum>:MATH:MEMorize

PNA COM Equivalent - Notes

DataToMemory Method	If the data is not in memory, store data into memory.
GetData Method	Output memory data.

8753 Command	Description	Range	Query Response
OUTPMSTA	Outputs the marker statistics in ASCII format: mean, standard deviation,	N/A	<num,num,num><LF

	and peak-to-peak variation in that order. If statistics is not on, it is turned on to generate current values and turned off again.		

PNA SCPI Equivalent - Notes

Step 1 CALC:FUNC:TYPE	Select the statistic TYPE that you can then query.
Step 2 CALC:FUNC:DATA?	Read the selected trace statistic.

PNA COM Equivalent - Notes

Get Trace Statistics Method	Returns the Trace Statistics.

8753 Command	Description	Range	Query Response
OUTPMWID	Outputs the marker bandwidths search results in ASCII format: bandwidth, center, and Q in that order. If widths is not on, it is turned on to generate current values and then turned off again.	N/A	<num,num,num><LF>
OUTPMWIL	Outputs the marker bandwidths search results in ASCII format: bandwidth, center, Q, and loss in that order. If widths is not on, it is turned on to generate current values and turned off again.	N/A	<num,num,num,num><LF>

PNA SCPI Equivalent - Notes

CALC:MARK:BWID	Use command to set and return filter statistics.

PNA COM Equivalent - Notes

Get Filter Statistics Method	Returns the Filter Statistics resulting from a SearchFilterBandwidth method.

8753 Command	Description	Range	Query Response
OUTPOPTS	Outputs an ASCII string of the options installed in the analyzer.	N/A	<$><LF

PNA SCPI Equivalent - Notes

*OPT?	Returns a string identifying the analyzer option configuration.

PNA COM Equivalent - Notes

Options Property	Returns a string identifying the analyzer option configuration.

8753 Command	Description	Range	Query Response
OUTPPRIN	Outputs a PCL raster dump of the display, intended for a graphics printer.	N/A	<$><LF

PNA SCPI Equivalent - Notes	
	No equivalent command at present.

PNA COM Equivalent - Notes	
DoPrint Method	Prints the screen to the active printer.
PrintToFile Method	Saves the screen data to a bitmap (.bmp) file.

8753 Command	Description	Range	Query Response
OUTPRAW	Outputs the selected raw data array.	Integers 14: 1=S11data 2=S21 data 3=S12 data 4=S22 data	<array><LF

PNA SCPI Equivalent - Notes	
CALC:DATA	Output the data array.

PNA COM Equivalent - Notes	
GetData Method	Output the data array.

8753 Command	Description	Range	Query Response
OUTPSERN	Outputs a string that contains the serial number of the analyzer.	N/A	<$><LF

PNA SCPI Equivalent - Notes	
*IDN?	Output the serial number.

PNA COM Equivalent - Notes	
IDString Property	Returns the ID of the analyzer, including the Model number, Serial Number, and the Software revision number.

8753 Command	Description	Range	Query Response
OUTPSTAT	Returns the status byte as an ASCII integer (0255) that can be interpreted as the 8-bit status byte. This command is the same as "STB?."	N/A	<num><LF

PNA SCPI Equivalent - Notes	
*STB?	Reads the value of the instrument status byte.

PNA COM Equivalent - Notes	
	No equivalent command at present.

8753 Command	Description	Range	Query Response
OUTPTITL	Outputs the display title in ASCII format.	N/A	<$><LF

PNA SCPI Equivalent - Notes	
DISP:WIND:TITL:DATA	Read-Write data in the window title area.

PNA COM Equivalent - Notes

Title Property	Writes or reads a custom title for the window.

8753 Command	Description	Range	Query Response
OUTPDATF	Fast data transfer command for OUTPDATA.	N/A	<array><LF
OUTPFORE	Fast data transfer command for OUTPFORM.	N/A	<array><LF
OUTPMEMF	Fast data transfer command for OUTPMEMO.	N/A	<array><LF
OUTPRAF<num>	Fast data transfer of the selected raw data array.	Integers 14: 1=S11data 2=S21 data 3=S12 data 4=S22 data	<array><LF

Notes

The PNA Series Network Analyzer outputs data at the fastest possible data rate at all times. Therefore, there are not any commands in the PNA Series that correspond to the above commands.

8753 Command	Description	Range	Query Response
OUTPACTI	Outputs the value of the active function, or the last active function if the active entry area is off. The value is returned in ASCII format.	N/A	<$><LF>
OUTPCHAN	Outputs the active channel number: 1, 2, 3, or 4.	N/A	<num><LF
OUTPDATP	Outputs the trace data indexed by point (see "SELPT").	N/A	<num,num><LF
OUTPDATR	Outputs the trace data for a range of points (see "SELMINPT," "SELMAXPT"). This is an ASCII (FORM4) transfer.	N/A	<array><LF

Notes

The PNA Series Network Analyzer has features to output data and state information much different than earlier network analyzers. Therefore, there are not any commands in the PNA Series that directly correspond to the above commands.

8753 Command	Description	Range	Query Response

OUTPAMAX	Outputs the max values for all limit line segments. This is an ASCII transfer (FORM4).	N/A	<array><LF
OUTPAMIN	Outputs the min values for all limit line segments. This is an ASCII transfer (FORM4).	N/A	<array><LF
OUTPLIMF	Outputs the limit test results for each failed point, followed by the number of failed points. This is an ASCII transfer.	N/A	<array><LF
OUTPSEGAF	Outputs the segment number and its limit test status for all active segments. This is an ASCII transfer.	N/A	<array><LF
OUTPSEGAM	Outputs the limit test min/max for all segments. Outputs the segment number, max stimulus, max value, min stimulus, min value for all active segments. This is an ASCII transfer.	N/A	<array><LF
OUTPSEGF	Outputs the limit test status for a specified segment. See also "SELSEG."	N/A	<0|1|-1><L F> Values returned for limit test status are: 0 (fail), 1 (pass), or -1 (no limit).
OUTPSEGM	Outputs limit test min/max for a specified segment. See also "SELSEG."	N/A	<num,num><LF
Notes		These limit and segment commands currently are not available.	

8753 Command	Description	Range	Query Response
OUTPERRO	Outputs the oldest error message in the error queue. Sends the error number first, and then the error message itself, as an ASCII (FORM4) string no longer than 50 characters.	N/A	<num,$><LF
OUTPFAIP	This command is	N/A	<array><LF

	similar to OUTPLIMF except that it reports the number of failures first, followed by the stimulus and trace values for each failed point in the test. ASCII format.		
OUTPICAL	Outputs the selected interpolated error coefficient array for the active cal on the active channel.	Two-digit integers 0112.	<array><LF
OUTPKEY	Outputs the key code of the last key pressed in ASCII format. An invalid key is reported with a 63, a knob turn with a -1. See programming manual for additional information.	N/A	<num><LF
OUTPLEAS	Outputs the learn string, which contains the entire front panel state, the limit table, and the list frequency table. It is always in binary format not intended for decoding.	N/A	<learnstring><LF
OUTPPLOT	Outputs the HP-GL plot string in ASCII format to the GPIB port. Can be directed to a plotter, or read into the computer.	N/A	<$><LF>
OUTPPMCAL	Outputs the power meter calibration array for channel 1 or channel 2. See programming manual for additional information.	Integers 1 or 2.	<array><LF
OUTPPRE	Outputs pre-raw data array <num>. See programming manual for additional information.	integers 14: 1=S11data 2=S21 data 3=S12 data 4=S22 data	<array><LF
OUTPPRNALL	Outputs all of the list values or the current page of operating parameters in ASCII format. See programming manual for additional	N/A	Rows of data separated by a <LF>. Ends with <LF><LF>.

	information.		
OUTPRFFR	Outputs the external source RF frequency. The instrument must be in external source mode, using either INSMEXSA or INSMEXSM.	N/A	<num><LF
OUTPSEQ	Outputs the specified sequence listing to the GPIB port.	Integers 16.	<$><LF>

Notes

These commands currently are not available.

PARA

8753 Command	Description	Range	Query Response
PARAOUT	Programs all GPIO output bits at once.	integers 0255	<num><LF

PNA SCPI Equivalent - Notes	
CONTrol:AUXiliary:C:DATA	Reads and writes a 4-bit value to Port C on the Aux I/O connector.

PNA COM Equivalent - Notes	
Put PortCData Method	Writes a 4-bit value to Port C on the Aux I/O connector (pins 22-25)

PHAO

8753 Command	Description	Range	Query Response
PHAO<num>	Sets the phase offset.	0360 degrees	<num><LF

PNA SCPI Equivalent - Notes	
CALC:CORR:OFFS:PHAS	Read-Write the phase offset for the selected measurement.

PNA COM Equivalent - Notes	
Phase Offset Property	Sets the Phase Offset.

PHAS

8753 Command	Description	Range	Query Response
PHAS	Selects the phase display format.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:FORM	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes	
Format Property	Sets (or returns) the display format of the measurement.

POIN

8753 Command	Description	Range	Query Response
POIN<num>	Sets the number of points in the sweep, or in a sweep	Choose from: 3, 11, 21, 26, 51, 101, 201, 401, 801, 1601	<num><LF

segment.

PNA SCPI Equivalent - Notes

SENS:SWE:POIN	Read-Write the number of data points for the measurement. (2 - 16001)

PNA COM Equivalent - Notes

Number of Points Property	Sets or returns the Number of Points. (2 - 16001)

POL

8753 Command	Description	Range	Query Response
POLA	Selects the polar display format.	N/A	<0|1>><LF
POLMLIN	Selects linear as the marker readout format for polar display.	N/A	<0|1>><LF
POLMLOG	Selects log as the marker readout format for polar display.	N/A	<0|1>><LF
POLMRI	Selects real/imaginary as the marker readout format for polar display.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:FORM	Selects the polar display format.
CALC:MARK	Use this command to turn on a marker.
CALC:MARK:FORM	Selects the appropriate marker readout format.

PNA COM Equivalent - Notes

Format Property	Selects the polar display format.
Marker Format Property	Selects the appropriate marker readout format.

PORE

8753 Command	Description	Range	Query Response
PORE<ON|OFF>	Turns port extensions on and off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

SENS:CORR:EXT	Read-Write port extensions ON or OFF.

PNA COM Equivalent - Notes

State Property	Turns port extensions ON or OFF.

PORT

8753 Command	Description	Range	Query Response
PORT1<num>[S]	Set the port extension length for Port 1	±10 seconds	<0|1>><LF
PORT2<num>[S]	Set the port extension length for Port 2	±10 seconds	<0|1>><LF
PORTA<num>[S]	Set the port extension length for	±10 seconds	<0|1>><LF

	Input A		
PORTB<num>[S]	Set the port extension length for Input B	±10 seconds	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:CORR:EXT:PORT	Read-Write the extension value at the specified port.
SENS:CORR:EXT:REC	Read-Write the extension value at the specified receiver.

PNA COM Equivalent - Notes	
Port1 Property	Sets the port extension value for Port 1.
Port2 Property	Sets the port extension value for Port 2.
InputA Property	Sets the port extension value for Receiver A.
InputB Property	Sets the port extension value for Receiver B.

PORTP

8753 Command	Description	Range	Query Response
PORTP<CPLD|UNCPLD>	Selects either coupled or uncoupled for the port powers of a given channel.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SOUR:POW:COUP	Read-Write Port Power Coupling ON or OFF.

PNA COM Equivalent - Notes	
CouplePorts Property	Turns ON and OFF source power coupling.

POWE

8753 Command	Description	Range	Query Response
POWE<num>[DB]	Sets the output power level.	output power range of your analyzer. The output power range of your analyzer depends upon the model and installed options. Refer to your analyzers users guide to determine the power range of your analyzer.	<num><LF

PNA SCPI Equivalent - Notes	
SOUR:POW	Read-Write the RF power output level.

PNA COM Equivalent - Notes	
Test Port Power Property	Read-Write the RF power output level.

POWL

8753 Command	Description	Range	Query Response
POWLFREQ	Selects the frequency for which a power loss correction is entered. This must be followed by a POWLLOSS<num>; command,which sets the value.	stimulus range	<num><L F >

POWLLOSS	Sets the loss value for a particular frequency, set by POWLFREQ, in the power loss list.	-9900 to 9900 dB	<num><L F >

PNA SCPI Equivalent - Notes	
SOUR:POW:CORR:COLL:TABL:FREQ	(Read-Write) Read or write frequency values for the selected table (cal factor table for a power sensor, or the loss compensation table).
SOUR:POW:CORR:COLL:TABL:DATA	(Read-Write) Read or write data into the selected table. If the selected table is a power sensor table, the data is interpreted as cal factors in units of percent. If the loss table is selected, the data is interpreted as loss in units of dB.

PNA COM Equivalent - Notes	
Frequency Property	Sets or returns the frequency associated with a PowerLossSegment.
CalFactor Property	Sets or returns the cal factor value associated with a power sensor cal factor segment.

POWR

8753 Command	Description	Range	Query Response
POWR<num>	Sets the source power range. See also "PRAN."	Use two-digit integers 0007.	N/A

PNA SCPI Equivalent - Notes	
SOUR:POW:ATT	Setting the attenuation is equivalent to setting the source power range.

PNA COM Equivalent - Notes	
Attenuator Property	Setting the attenuation is equivalent to setting the source power range.

POWS

8753 Command	Description	Range	Query Response
POWS	Selects power sweep from the sweep type menu.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
SENS:SWE:TYPE	Read-Write the type of analyzer sweep mode.

PNA COM Equivalent - Notes	
Sweep Type Property	Sets the type of X-axis sweep that is performed on a channel.

POWT

8753 Command	Description	Range	Query Response
POWT<ON|OFF>	Sets source power on or off. Works the opposite of the SOUP command. Sending POWTON turns source power off.	N/A	<0|1>><LF>

	Sending POWTOFF turns source power on.		

PNA SCPI Equivalent - Notes

OUTP	Turns ON and OFF Source Power.

PNA COM Equivalent - Notes

Source Power State Property	Turns ON and OFF Source Power.

PRAN

8753 Command	Description	Range	Query Response
PRAN<num>	Sets the source power range. See also "POWR."	integers 07.	N/A

PNA SCPI Equivalent - Notes

SOUR:POW:ATT	Setting the attenuation is equivalent to setting the source power range.

PNA COM Equivalent - Notes

Attenuator Property	Setting the attenuation is equivalent to setting the source power range.

PRES

8753 Command	Description	Range	Query Response
PRES	Presets the analyzer to the factory preset state. OPC-compatible.	N/A	N/A

PNA SCPI Equivalent - Notes

SYST:PRES	Preset.

PNA COM Equivalent - Notes

Preset Method	Preset

PRIN

8753 Command	Description	Range	Query Response
PRINALL	Copies the display, in raster graphics mode, to a printer. Requires pass control when using the GPIB port. (Use PRINTALL to send ASCII data to the printer.)	N/A	N/A

PNA SCPI Equivalent - Notes

	No equivalent command at present.

PNA COM Equivalent - Notes

Do Print Method	Prints the screen to the active printer.
PrintToFile Method	Saves the screen data to a bitmap (.bmp) file.

PWMC

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>PWMCONES</td><td>Power Meter Cal done on one sweep. A calibration sweep should be taken. a</td><td>-100dB to 100dB</td><td><0|1>><LF></td></tr>
</table>

	calibration sweep should be taken (TAK) to ensure a valid power calibration.		

PNA SCPI Equivalent - Notes	
	Selects the source power calibration method.
PNA COM Equivalent - Notes	

PWRR

8753 Command	Description	Range	Query Response
PWRR<PMAN|PAUTO>	Selects whether the power range is in auto or manual mode.	N/A	<0|1>><LF 0 = manual mode; 1 = auto mode

PNA SCPI Equivalent - Notes	
SOUR:POW:ATT:AUTO	Read-Write automatic attenuation control ON or OFF. Setting the automatic attenuation control is equivalent to setting the source power range mode.
PNA COM Equivalent - Notes	
Attenuator Mode Property	Sets or returns the mode of operation of the attenuator control. Setting the automatic attenuation control is equivalent to setting the source power range mode.

PWRLOSS

8753 Command	Description	Range	Query Response
PWRLOSS	Selects whether or not to use the power loss table for a power meter calibration	N/A	<0|1>><LF>

PNA SCPI Equivalent - Notes	
SOUR:POW:CORR:COLL:TABL:LOSS	(Read-Write) Indicates whether or not to adjust the power readings using the values in the loss table during a source power cal sweep.
PNA COM Equivalent - Notes	
UsePowerLossSegments Property	Specifies if subsequent calls to the AcquirePowerReadings method will make use of the loss table (PowerLossSegments).

RAI

8753 Command	Description	Range	Query Response
RAID	Completes the response and isolation cal sequence. OPC-compatible.	N/A	N/A
RAIISOL	Calls the isolation class for the	N/A	N/A

	response and isolation calibration.		
RAIRESP	Calls the response class for the response and isolation calibration.	N/A	N/A

PNA SCPI Equivalent - Notes	
SENS:CORR:COLL:SAVE	Write-only to calculate the correction data using the selected :METHod and turn error correction ON.
SENS:CORR:COLL	Write-only to measure the specified standard from the selected calibration kit.
SENS:CORR:COLL:METH	Read-Write the calibration method.

PNA COM Equivalent - Notes	
Calculate Error Coefficients Method	Calculates the correction data using the selected Cal Type and turns error correction ON.
AcquireCalStandard2 Method	Measures the specified standard from the selected calibration kit.
SetCalInfo Method	Specifies the type of calibration to **perform**.

REAL

8753 Command	Description	Range	Query Response
REAL	Sets the display format to real.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:FORM	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes	
Format Property	Sets (or returns) the display format of the measurement.

REF

8753 Command	Description	Range	Query Response
REFP<num>	Enters the reference position. 0 is the bottom, 10 is the top of the graticule.	Integers 0–10	<num><LF
REFV<num>	Enters the reference line value.	Amplitude range. For log mag: ± 500 dB. For phase: ± 500 degrees. For Smith chart and Polar: ± 500 units. For linear magnitude: ± 500 units. For SWR: ± 500 units. The scale is always positive, and has minimum values of 0.001dB, 10e-12 degrees, 10e-15 seconds, and 10 picounits.	<num><LF

PNA SCPI Equivalent - Notes

DISP:WIND:TRAC:Y:RPOS	Read-Write the **Reference Position** of the specified trace in the specified window.
DISP:WIND:TRAC:Y:RLEV	Read-Write the Y axis **Reference Level** of the specified trace in the specified window.
PNA COM Equivalent - Notes	
Reference Position Property	Sets or returns the **Reference Position** of the active trace.
Reference Value Property	Sets or returns the value of the Y-axis **Reference Level** of the active trace.

REIC

8753 Command	Description	Range	Query Response
REIC	Sets the power level reference value for a power calibration	Amplitude Range	N/A

PNA SCPI Equivalent - Notes

CALC:CORR:OFFS	(Read-Write) Specifies the power level to which the selected (unratioed) measurement's data is to be adjusted by a Receiver Power Calibration. This command applies only when the selected measurement is of unratioed power.
PNA COM Equivalent - Notes	
LogMagnitudeOffset Property	Sets or returns the power offset value in dBm that the normalized unratioed power measurement data will be shifted by. The unratioed power measurement is effectively calibrated to the power level specified by the value of LogMagnitudeOffset as soon as the Normalization property is set to ON after the DataToDivisor method has been called.

RESPDONE

8753 Command	Description	Range	Query Response
RESPDONE	Completes the response calibration sequence. OPC-compatible.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:CORR:COLL:SAVE	Write-only to calculate the correction data.
PNA COM Equivalent - Notes	
Calculate Error Coefficients Method	Calculates the correction data.

REST

8753 Command	Description	Range	Query Response
REST	Measurement restart.	N/A	N/A

PNA SCPI Equivalent - Notes

Step 1 ABOR	Abort the current sweep with the command in Step 1.
Step 2 INIT	Initiate a new sweep with the command in Step 2.
PNA COM Equivalent - Notes	
	No equivalent command at present.

REV

8753 Command	Description	Range	Query Response
REVI	Calls the reverse isolation calibration class during a full 2-port calibration.	N/A	N/A
REVM	Calls the reverse match calibration class during a full 2-port calibration.	N/A	N/A
REVT	Calls the reverse transmission calibration class during a full 2-port calibration.	N/A	N/A

PNA SCPI Equivalent - Notes	
SENS:CORR:COLL:SAVE	Write-only to calculate the correction data using the selected :METHod and turn error correction ON.
SENS:CORR:COLL	Write-only to measure the specified standard from the selected calibration kit.
SENS:CORR:COLL:METH	Read-Write the calibration method.

PNA COM Equivalent - Notes	
Calculate Error Coefficients Method	Calculates the correction data using the selected Cal Type and turns error correction ON.
AcquireCalStandard2 Method	Measures the specified standard from the selected calibration kit.
SetCalInfo Method	Specifies the type of calibration to **perform**.

RST

8753 Command	Description	Range	Query Response
RST	Presets the analyzer to the factory preset state. OPC-compatible.	N/A	N/A

PNA SCPI Equivalent - Notes	
*RST	Executes a device reset and cancels any pending *OPC command or query. Replace "RST" with "*RST".

PNA COM Equivalent - Notes	
Reset Method	Resets instrument. Clears all existing windows and measurements.

S

8753 Command	Description	Range	Query Response
S11	Forward reflection measurement.	N/A	<0|1>><LF
S12	Reverse transmission measurement.	N/A	<0|1>><LF
S21	Forward transmission	N/A	<0|1>><LF

	measurement.		
S22	Reverse reflection measurement.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

	Follow the steps below to create and display a measurement.
Step 1 CALC:PAR:DEF	Create the measurement.
Step 2 DISP:WIND	If a new window will be used to display the measurement, then create a window.
Step 3 DISP:WIND:TRAC:FEED	Display the measurement in the window.

PNA COM Equivalent - Notes

CreateMeasurement Method	Create and display the measurement.

SADD

8753 Command	Description	Range	Query Response
SADD	Adds a new segment to the table during a list-frequency, limit-table, cal sensor table, or power loss table edit.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:SEGM:ADD	Write-only to add a segment. A segment must be added prior to setting data in the segment.

PNA COM Equivalent - Notes

Add segments Method	Add a segment.

SCAL

8753 Command	Description	Range	Query Response
SCAL<num>	Sets the trace scale factor.	Amplitude range. For log mag: ± 500 dB. For phase: ± 500 degrees. For Smith chart and Polar: ± 500 units. For linear magnitude: ± 500 units. For SWR: ± 500 units. The scale is always positive, and has minimum values of 0.001dB, 10e-12 degrees, 10e-15 seconds, and 10 picounits.	<num><LF

PNA SCPI Equivalent - Notes

DISP:WIND:TRAC:Y:PDIV	Read-Write the Y axis **Per Division** value of the specified trace in the specified window.

PNA COM Equivalent - Notes

YScale Property	Sets or returns the Y-axis **Per Division** value of the active trace.

SDEL

8753 Command	Description	Range	Query Response

SDEL	Deletes the current segment while editing a list frequency, a limit table, or a power loss list.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:SEGM:DEL	Write-only to delete the specified segment number.
SENS:SEGM:DEL:ALL	Write-only to delete all segments.
CALC:LIM:DATA	Limit lines always remain in memory. Use this SCPI command to set limit segment OFF.

PNA COM Equivalent - Notes

Remove Method	Removes an item from a collection of objects.

SEA

- SEAL
- SEAMAX
- SEAMIN
- SEAOFF
- SEAR
- SEATARG

8753 Command	Description	Range	Query Response
SEAL	Search left for next occurrence of the target value.	N/A	N/A
SEAR	Search right for next occurrence of the target value.	N/A	N/A

PNA SCPI Equivalent - Notes

Step 1 CALC:MARK:TARG	Read-Write the target value for the specified marker when doing Target Searches.
Step 2 CALC:MARK:FUNC:EXEC	Write-only to immediately execute (perform) the specified search function.

PNA COM Equivalent - Notes

Step 1 TargetValue Property	Sets the target value for the marker when doing Target Searches.
Step 2a Search Target Left Method	Moving to the left of the marker position, searches the marker's domain for the target value.
Step 2b Search Target Right Method	Moving to the right of the marker position, searches the marker's domain for the target value.

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>SEAMAX</td><td>Search for trace maximum on the current channel.</td><td>N/A</td><td><0|1><LF></td></tr>
<tr><td>SEAMIN</td><td>Search for trace minimum on the</td><td>N/A</td><td><0|1><LF></td></tr>
</table>

current channel.

PNA SCPI Equivalent - Notes

CALC:MARK:FUNC:EXEC	Write-only to immediately execute (perform) the specified search function.

PNA COM Equivalent - Notes

Search Max Method	Searches the marker domain for the maximum value.
Search Min Method	Searches the marker domain for the minimum value.

8753 Command	Description	Range	Query Response
SEAOFF	Turns the marker search off.	N/A	<0|1><LF>

PNA SCPI Equivalent - Notes

CALC:MARK	Turn marker search off by turning OFF the marker.

PNA COM Equivalent - Notes

Delete Marker Method	Turn marker search off by turning OFF the marker.

8753 Command	Description	Range	Query Response
SEATARG<num>	Set the search target amplitude.	Amplitude range.	<num><LF>

PNA SCPI Equivalent - Notes

CALC:MARK:TARG	Sets the target value for the marker when doing Target Searches.

PNA COM Equivalent - Notes

TargetValue Property	Sets the target value for the marker when doing Target Searches.

SEDI

8753 Command	Description	Range	Query Response
SEDI<num>	During either a frequency, limit, or power loss table edit, selects segment <num> for editing.	State dependent. Range for frequency segment = 1 to 30; Range for limit test segment = 1 to 18; Range for power loss table segment = 1 to 12	<num><LF>

PNA SCPI Equivalent - Notes

	PNA Network Analyzers allow one to directly edit a segment or limit line. To edit a segment or limit line, see the following commands:
Sense:Segment	Commands to edit a segment.
Calc:Limit	Commands to edit a limit line.

PNA COM Equivalent - Notes

PNA Network Analyzers allow one to directly edit a segment or limit line. See the appropriate methods and properties for segments and limit lines.

SEG

8753 Command	Description	Range	Query Response
SEGIFBW<num>	Sets the IFBW for the active segment of a list-frequency table in swept list mode.	Choose from 10, 30, 100, 300, 1000, 3000, 3700, 6000.	see "Note" below
SEGPOWER<num>	Sets the power for the active segment of a list-frequency table in swept list mode.	Output power range of your analyzer. The output power range is dependent upon the model and option configuration of your analyzer. Refer to your analyzers users guide to determine the output power range of your analyzer.	see "Note" below
	Note: Currently these commands can be queried by sending the command followed by the OUTPACTI command.		

PNA SCPI Equivalent - Notes

SENS:SEGM:BWID	Read-Write the IFBandwidth for the specified segment.
SENS:SEGM:POW	Read-Write the Port Power level for the specified segment.

PNA COM Equivalent - Notes

IF Bandwidth Property	Sets or returns the IF Bandwidth of all measurements in a channel. **OR** Sets or returns the IF Bandwidth of a specified sweep segment.
Test Port Power Property	Sets or returns the RF power level of all measurements in a channel **or** Sets or returns the RF power level of a specified sweep segment.

SING

8753 Command	Description	Range	Query Response
SING	Single sweep. OPC-compatible.	N/A	N/A

PNA SCPI Equivalent - Notes

INIT:CONT	If sweep is not is single sweep mode, put the analyzer in single sweep mode by setting continuous OFF.
INIT	Trigger one sweep.

PNA COM Equivalent - Notes

Single_Method	Single sweep.

SMI

8753 Command	Description	Range	Query Response
SMIC	Selects Smith chart display format.	N/A	<0|1>><LF
SMIMGB	Selects G+jB (conductance and susceptance) marker readout on a Smith chart.	N/A	<0|1>><LF
SMIMLIN	Selects linear magnitude marker readout on a Smith chart.	N/A	<0|1>><LF
SMIMLOG	Selects log magnitude marker readout on a Smith chart.	N/A	<0|1>><LF
SMIMRI	Selects real/imaginary pairs (resistance and reactance) marker readout on a Smith chart.	N/A	<0|1>><LF
SMIMRX	Selects R + jX marker readout on a Smith chart.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:FORM	Selects the Smith chart display format.
CALC:MARK	Use this command to turn on a marker.
CALC:MARK:FORM	Selects the appropriate marker readout format.

PNA COM Equivalent - Notes	
Format Property	Selects the Smith chart display format.
Marker Format Property	Selects the appropriate marker readout format.

SMOO

8753 Command	Description	Range	Query Response
SMOOAPER<num>	Sets the smoothing aperture as a percent of the trace.	0.05 to 20%	<num><LF
SMOOO<ON|OFF>	Selects whether smoothing is on or off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes	
CALC:SMO:APER	Read-Write the amount of smoothing.
CALC:SMO	Read-Write data smoothing ON or OFF.

PNA COM Equivalent - Notes	
Smoothing Aperture Property	Specifies or returns the amount of smoothing.
Smoothing Property	Turns data smoothing ON and OFF.

SOUP

8753 Command	Description	Range	Query Response

SOUP<ON|OFF>	Selects whether the source power is on or off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

OUTP	Read-Write RF power from the source ON or OFF.

PNA COM Equivalent - Notes

Source Power State Property	Turns source power ON and OFF.

SPAN

8753 Command	Description	Range	Query Response
SPAN<num>[HZ|DB]	Sets the stimulus span value. If a list frequency segment is being edited, sets the span of the list segment.	Stimulus range. For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ± 1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes

SENS:FREQ:SPAN	Read-Write the frequency span of the analyzer.
SENS:SEGM:FREQ:SPAN	Read-Write the frequency span **for the specified segment**.

PNA COM Equivalent - Notes

Frequency Span Property	Sets or returns the frequency span of all measurements in a channel **or** Sets or returns the frequency span of a specified sweep segment.

SRE

8753 Command	Description	Range	Query Response
SRE<num>	Service request enable. A bit set in <num> enables the corresponding bit in the status byte to generate an SRQ.	integers 0255	<num><LF

PNA SCPI Equivalent - Notes

*SRE	Enables bits in the service request register. Replace "SRE" with "*SRE".

PNA COM Equivalent - Notes

No equivalent command at present.

SSEG

8753 Command	Description	Range	Query Response

SSEG<num>	Selects the desired segment of the frequency list for a list frequency sweep. See also "ASEG".	Integers 130	<num><LF

PNA SCPI Equivalent - Notes

SENS:SEGM	Read-Write the specified segment ON or OFF.
SENS:SWE:TYPE	The segment will not be turned on until the sweep type is set to "SEGMent" sweep with this command.

PNA COM Equivalent - Notes

Segments Collection	Segment collection object.

STAR

8753 Command	Description	Range	Query Response
STAR<num>[HZIDB]	Sets the start stimulus value. If a list frequency segment is being edited, sets the start of the list segment.	Stimulus range. For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ± 1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes

SENS:SWE:TYPE	Read-Write the start frequency of the analyzer.
SENS:SEGM:FREQ:STAR	Read-Write the start frequency **for the specified segment**.

PNA COM Equivalent - Notes

Start Frequency Property	Sets or returns the start frequency of all measurements in a channel **or** Sets or returns the start frequency of a specified sweep segment.

STB?

8753 Command	Description	Range	Query Response
STB?	Query only. Outputs the status byte in ASCII format (FORM4). Same as OUTPSTAT.	N/A	<num><LF

PNA SCPI Equivalent - Notes

*STB?	Enables bits in the service request register. Replace "STB?" with "*STB?".

PNA COM Equivalent - Notes

No equivalent command at present.

STOP

8753 Command	Description	Range	Query Response
STOP<num>[HZ|DB]	Sets the stop stimulus value. If a list frequency segment is being edited, sets the stop of the list segment.	Stimulus range. For frequency or power sweeps, refer to "Preset State and Memory Allocation," in your analyzers users guide. For CW time: 0 to 24 hours. For frequency sweep, transform on: ± 1/frequency step. For CW time sweep, transform on: ±1/time step.	<num><LF

PNA SCPI Equivalent - Notes	
SENS:FREQ:STOP	To Read-Write the stop frequency of the analyzer.
SENS:SEGM:FREQ:STOP	To Read-Write the stop frequency **for the specified segment**.

PNA COM Equivalent - Notes	
Stop Frequency Property	Sets or returns the stop frequency of all measurements in a channel **or** Sets or returns the stop frequency of a specified sweep segment.

STOR

8753 Command	Description	Range	Query Response
STOR<num>	Stores the current instrument state to disk using the file name provided by the preceding TITF<num> command.	Integers 15	N/A

PNA SCPI Equivalent - Notes	
MMEM:STOR:STAT	Write-only to store the specified file.

PNA COM Equivalent - Notes	
Save Method	Saves a measurement state, calibration state, or both.

SWE

8753 Command	Description	Range	Query Response
SWEA	Automatically selects the fastest sweep time based on the current analyzer settings for number of points, IF bandwidth, sweep mode, averaging condition	N/A	N/A

	and frequency span.		
SWET<num>[S]	Sets the sweep time. (Setting SWET0 is equivalent to sending the SWEA command.)	086,400 s	<num><LF

PNA SCPI Equivalent - Notes

SENS:SWE:TIME:AUTO	Read-Write the automatic sweep time function ON or OFF.
SENS:SWE:TIME	Read-Write the time the analyzer takes to complete one sweep.

PNA COM Equivalent - Notes

Sweep Time Property	Sets the Sweep time of the analyzer. Setting sweep time to 0 will result in the fastest possible sweep time with the current settings.

SWR

8753 Command	Description	Range	Query Response
SWR	Selects the SWR display format.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:FORM	Read-Write the display format for the measurement.

PNA COM Equivalent - Notes

Format Property	Sets (or returns) the display format of the measurement.

TALKLIST

8753 Command	Description	Range	Query Response
TALKLIST	Selects the talker listener mode.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

	No equivalent command at present.

PNA COM Equivalent - Notes

GPIBMode Property	Selects the talker listener mode.

TAK

8753 Command	Description	Range	Query Response
TAKRS	Begins a receiver calibration sweep	N/A	N/A

PNA SCPI Equivalent - Notes

CALC:NORM:IMM	Stores the selected measurements data to that measurements divisor buffer for use by the Normalization data processing algorithm. This command is not compatible with ratioed measurements such as S-parameters. It is intended for receiver power calibration when the selected measurement is of an unratioed power type.

PNA COM Equivalent - Notes

DataToDivisor Method	Stores the measurements data to the

measurements divisor buffer for use by the Normalization data processing algorithm. Normalization is currently supported only on measurements of unratioed power, for purpose of receiver power calibration.

TIT

8753 Command	Description	Range	Query Response
TITL<$>	Enters a new display title.	48 characters max	N/A

PNA SCPI Equivalent - Notes

DISP:WIND:TITL:DATA — Read-Write data in the window title area.

PNA COM Equivalent - Notes

Title Property — Writes or reads a custom title for the window.

8753 Command	Description	Range	Query Response
TITF	Titles the indicated file numbers.	**<num>:** 15 **<$>:** 10 char. max.	N/A
TITP	Titles the plot to disk file.	10 characters max	N/A
TITR	Titles the indicated internal register.	**<num>:** 15 **<$>:** 10 char. max.	N/A
TITREG	Titles save/recall registers 01 through 31. TITREG01 through TITREG05 are the same as TITR1 through TITR5.	**<num>:** 0131 **<$>:** 10 char. max.	N/A
TITSEQ	Selects the sequence to be titled.	**<num>:** 16 **<$>:** 10 char. max.	N/A
TITSQ	Provides access to the sequence title functions.	N/A	N/A

Notes

These commands currently are not available on PNA

TRACK

8753 Command	Description	Range	Query Response
TRACK<ON|OFF>	Turns marker search tracking on and off.	N/A	<0|1>><LF

PNA SCPI Equivalent - Notes

CALC:MARK:FUNC:TRAC — Read-Write tracking capability for the specified marker.

PNA COM Equivalent - Notes

Tracking Property — Turns marker search tracking on and off.

TRL

8753 Command	Description	Range	Query Response
TRLL1	Measures TRL Line/match for Port 1 during a TRL/LRM 2-port calibration.	N/A	N/A
TRLL2	Measures TRL Line/match for Port 2 during a TRL/LRM 2-port calibration.	N/A	N/A
TRLR1	Measures TRL S11 reflect during a TRL/LRM 2-port calibration.	N/A	N/A
TRLR2	Measures TRL S22 reflect during a TRL/LRM 2-port calibration.	N/A	N/A
TRLT	Measures TRL thru during a TRL/LRM 2-port calibration.	N/A	N/A

PNA SCPI Equivalent - Notes	
SENS:CORR:COLL:METH	Read-Write the calibration method.
SENS:CORR:COLL	Write-only to measure the specified standard from the selected calibration kit.

PNA COM Equivalent - Notes	
SetCalInfo_Method	Specifies the type of calibration to **perform**.
AcquireCalStandard2 Method	Measures the specified standard from the selected calibration kit.

TST?

8753 Command	Description	Range	Query Response
TST?	Query only. Causes a self test and returns a zero if the test is passed.	N/A	<num><LF>

PNA SCPI Equivalent - Notes	
*TST?	Returns the result of a complete self-test. An ASCII 0 indicates no failures found.

PNA COM Equivalent - Notes	
	No equivalent command at present.

TSTP

8753 Command	Description	Range	Query Response
TSTP<P1|P2>	Selects test port 1 or 2 for non-S-parameter measurements.	N/A	N/A

PNA SCPI Equivalent - Notes

SENS:SWE:SRCP — Read-Write the source port when making non S-parameter measurements. Has no effect on S-parameter measurements.

PNA COM Equivalent - Notes

CreateMeasurement Method — Create and display the measurement. Method parameter allows one to select the specific port.

TTL

<table>
<tr><th>8753 Command</th><th>Description</th><th>Range</th><th>Query Response</th></tr>
<tr><td>TTLLPULS</td><td>TTL normally high, low pulse at end of sweep.</td><td>N/A</td><td><0|1><L F ></td></tr>
<tr><td>TTLHPULS</td><td>TTL normally low, high pulse at end of sweep.</td><td>N/A</td><td><0|1><L F ></td></tr>
</table>

PNA SCPI Equivalent - Notes

CONT:AUX:SWE — (Read-Write) Specifies the event that will cause the AUX IO Sweep End line (pin 11) to go to a low (false) state. The line will return to a high state after the appropriate calculations are complete. This line is connected internally to the Sweep End line of the Material Handler IO.

PNA COM Equivalent - Notes

SweepEndMode Property — (Read-Write) Specifies the event that will cause the AUX IO Sweep End line (pin 11) to go to a low (false) state. The line will return to a high state after the appropriate calculations are complete. This line is connected internally to the Sweep End line of the Material Handler IO.

USESENS

8753 Command	Description	Range	Query Response
USESENSA USESENSB	Selects the power meter input being used for a power calibration	N/A	N/A

PNA SCPI Equivalent - Notes

SOUR:POW:CORR:COLL — Initiates a source power cal acquisition sweep using the power sensor attached to the specified channel (A or B) on the power meter.

PNA COM Equivalent - Notes

AcquirePowerReadings Method — Initiates a source power cal acquisition

---

VELOFACT

8753 Command	Description	Range	Query Response
VELOFACT<num>	Enters the velocity factor of the transmission medium.	0 to 10	<num><LF

PNA SCPI Equivalent - Notes

SENS:CORR:RVEL:COAX	Read-Write the velocity factor to be used with Electrical Delay and Port Extensions.

PNA COM Equivalent - Notes

Velocity Factor Property	Sets the velocity factor to be used with Electrical Delay and Port Extensions.

WAIT

8753 Command	Description	Range	Query Response
WAIT	Waits for a clean sweep when used with the OPC command.	N/A	N/A

PNA SCPI Equivalent - Notes

*WAI	Prohibits the instrument from executing any new commands until all pending overlapped commands have been completed.

PNA COM Equivalent - Notes

No equivalent command at present.

WID

8753 Command	Description	Range	Query Response
WIDT<ON|OFF>	Turns the bandwidth search on and off.	N/A	<0|1><LF
WIDV<num>	Enters the widths search parameter.	Amplitude range.	<num><LF

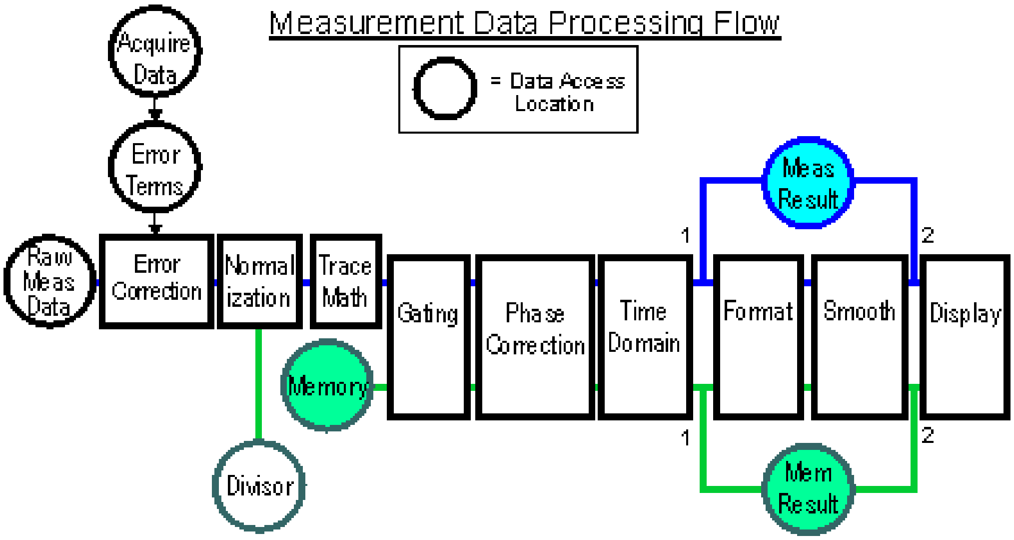
PNA SCPI Equivalent - Notes

Step 1 CALC:MARK:BWID	Turn ON bandwidth search. Also, can return statistics.
Step 2 CALC:MARK:FUNC:TRAC	Turn marker tracking ON.
Step 3 CALC:MARK:AOFF	Turn OFF bandwidth search. This is accomplished by turning all markers OFF. Markers can also be turned OFF one at a time with CALCulate<cnum>:MARKer<mkr>[:STATe] <ON|OFF>

PNA COM Equivalent - Notes

Step 1 Bandwidth Target Property	Enter the bandwidth target value.
Step 2 Search Filter Bandwidth Method	Turn ON bandwidth search.
Step 3 Tracking Property	Turn marker tracking ON.
Get Filter Statistics Method	The filter statistics can be returned with this method.
DeleteAllMarkers Method	Turn OFF bandwidth search. This is accomplished by turning all markers OFF. Markers can also be turned OFF one at a time

with DeleteMarker method.



Click a box to view process details:

Raw Measurement Data - Complex trace data which is ratioed if required by the parameter, such as S11 or A/B. Otherwise it is raw receiver data, such as A or B. This data is averaged if Averaging is ON. See Measurement Parameters

---

Standard Acquisition Data - Raw Complex Data resulting from measuring calibration standards or recalling a calibration. See Measurement Calibration.

Error Term Data - Data that is calculated from Acquisition data using formulas which are appropriate for the selected calibration method.

Error Correction - Error terms are applied to the raw measurement data if error correction is ON. Otherwise this data is identical to Raw Measurement Data.

Divisor - Correction data resulting from a Receiver power calibration. See Receiver power calibration

---

Normalization - If performing Receiver power correction, applies the "Divisor" correction data to the measurement.

Trace Math - If turned ON, memory data is combined with measurement data using the selected math function. Available functions are: Data+Mem, Data-Mem, Data*Mem, and Data/Mem. See Math Operations.

Memory Data - Complex trace data resulting from a Data-To-Memory operation. Each measurement can have one memory trace. The memory data parallels the measurement data through the remaining post processing blocks. For example, turning smoothing ON will smooth both the measurement and memory traces.

Gating - If turned ON, Filter Gating is applied to the measurement data. The gates are used to select regions of the trace where a subsequent transform will be applied. See Gating.

Phase Correction - If turned ON, applies electrical delay, phase offset, and port extensions. These are all separate features that are controlled individually. See Phase Measurement Accuracy.

Time Domain - If turned ON, transforms the data from the frequency domain to the time domain. See Time Domain

Formatter - Complex data is converted into scalar data formats for screen display and remote access. For smoothed data, request the data in the same format as the displayed data. See Data Format

Memory Result Data- Memory data is formatted and available for remote access from access point 1. To get smoothed data, request the data in the same format as the displayed data. The data will then come from access point 2.

Measurement Result Data - Measurement data is formatted and available for remote access from access point 1. To get smoothed data, request the data in the same format as the displayed data. The data will then come from access point 2.

Smoother - If turned ON, removes discontinuities in the measurement and memory trace. See Smoothing.

Display - Displays the processed measurement and / or memory data in the format of your choice. If remotely requested data is the same format as the displayed data, the requested data comes from this buffer.

Input I LO I IF I Output

Description	**SCPI**	**COM**
Loads a previously-configured mixer attributes file (.mix)	MIX:LOAD	LoadFile
Saves the settings for the mixer/converter test setup to a mixer attributes file.	MIX:SAVE	SaveFile
Number of Stages (1 or 2)	MIX:STAGE	LOStage
Set Input		
Start frequency value of the mixer input	MIX:SENS:INP:FREQ:STAR	InputStartFrequency
Stop frequency value of the mixer input	MIX:SENS:INP:FREQ:STOP	InputStopFrequency

Input Power level	MIX:SENS:INP:POW	InputPower
Numerator value of the Input Fractional Multiplier.	MIX:SENS:INP:FREQ:NUM	InputNumerator
Denominator value of the Input Fractional Multiplier.	MIX:SENS:INP:FREQ:DEN	InputDenominator
Set LO		
LO name		LOName
LO frequency value	MIX:SENS:LO:FREQ:FIX	LOFixedFrequency
LO Power	MIX:SENS:LO:POW	LOPower
Numerator value of the LO Fractional Multiplier.	MIX:SENS:LO:FREQ:NUM	LONumerator
Denominator value of the LO Fractional Multiplier.	MIX:SENS:LO:FREQ:DEN	LODenominator
Set IF		
Sideband (high or low)	MIX:SENS:IF:FREQ:SIDE	IFSideband
Start frequency value of the mixer IF.	MIX:SENS:IF:FREQ:STAR	IFStartFrequency
Stop frequency value of the mixer IF.	MIX:SENS:IF:FREQ:STOP	IFStopFrequency
Numerator value of the IF Fractional Multiplier.	MIX:SENS:IF:FREQ:NUM	IFNumerator
Denominator value of the IF Fractional Multiplier.	MIX:SENS:IF:FREQ:DEN	IFDenominator
Set Output		
Output sideband value (high or low)	MIX:SENS:OUT:FREQ:SIDE	OutputSideband
Start frequency value of the mixer output	MIX:SENS:OUT:FREQ:STAR	OutputStartFrequency
Stop frequency value of the mixer output	MIX:SENS:OUT:FREQ:STOP	OutputStopFrequency
Numerator value of the output Fractional Multiplier.	MIX:SENS:OUT:FREQ:NUM	OutputNumerator
Denominator value of the output Fractional Multiplier.	MIX:SENS:OUT:FREQ:DEN	OutputDenominator