

# **Programming Information**

*(Printed Version of Help)*

## **Agilent Technologies**

### **PNA Series Network Analyzers**



**Part Number: E8356-90028**  
**Printed in USA**  
**September 2002**

Supersedes June 2002

©Copyright 2000–2002 Agilent Technologies



<b><u>COM VERSUS SCPI</u></b> .....	5
<b><u>COMMAND FINDER</u></b> .....	6
<b><u>PNA OBJECT MODEL</u></b> .....	16
<b><u>OBSOLETE COMMANDS</u></b> .....	17
<b><u>APPLICATION OBJECT</u></b> .....	18
<b><u>COLLECTION METHODS AND PROPERTIES</u></b> .....	52
<b><u>CALIBRATOR OBJECT</u></b> .....	56
<b><u>CALFACTORSEGMENTS COLLECTION</u></b> .....	72
<b><u>CAL SET OBJECT</u></b> .....	73
<b><u>CAL SETS COLLECTION</u></b> .....	90
<b><u>CALKIT OBJECT</u></b> .....	90
<b><u>CALMANAGER OBJECT</u></b> .....	94
<b><u>CALSTANDARD OBJECT</u></b> .....	97
<b><u>CHANNEL OBJECT</u></b> .....	106
<b><u>CHANNELS COLLECTION</u></b> .....	130
<b><u>GATING OBJECT</u></b> .....	130
<b><u>HWAUXIO OBJECT</u></b> .....	134
<b><u>HWEXTERNALTESTSETIO OBJECT</u></b> .....	141
<b><u>HWMATERIALHANDLERIO OBJECT</u></b> .....	145
<b><u>IARRAYTRANSFER INTERFACE</u></b> .....	151
<b><u>IARRAYTRANSFER2 INTERFACE</u></b> .....	157
<b><u>ICALDATA INTERFACE</u></b> .....	164
<b><u>ICALDATA2 INTERFACE</u></b> .....	170
<b><u>ICALIBRATOR2 INTERFACE</u></b> .....	172
<b><u>INACUSTOMMEASUREMENT INTERFACE</u></b> .....	173

<b><u>ISOURCEPOWERCALDATA INTERFACE</u></b> .....	<b>175</b>
<b><u>LIMIT TEST COLLECTION</u></b> .....	<b>177</b>
<b><u>LIMITSEGMENT OBJECT</u></b> .....	<b>179</b>
<b><u>MARKER OBJECT</u></b> .....	<b>182</b>
<b><u>MEASUREMENTS COLLECTION</u></b> .....	<b>196</b>
<b><u>MEASUREMENT OBJECT</u></b> .....	<b>197</b>
<b><u>NAWINDOWS COLLECTION</u></b> .....	<b>223</b>
<b><u>NAWINDOW OBJECT</u></b> .....	<b>224</b>
<b><u>PORT EXTENSION OBJECT</u></b> .....	<b>230</b>
<b><u>POWERLOSSSEGMENTS COLLECTION</u></b> .....	<b>232</b>
<b><u>POWERLOSSSEGMENT OBJECT</u></b> .....	<b>233</b>
<b><u>POWERSENSOR OBJECT</u></b> .....	<b>235</b>
<b><u>POWERSENSORCALFACTORSEGMENT OBJECT</u></b> .....	<b>238</b>
<b><u>POWERSENSORS COLLECTION</u></b> .....	<b>239</b>
<b><u>SCPISTRINGPARSER OBJECT</u></b> .....	<b>239</b>
<b><u>SEGMENTS COLLECTION</u></b> .....	<b>240</b>
<b><u>SEGMENT OBJECT</u></b> .....	<b>242</b>
<b><u>SOURCEPOWERCALIBRATOR OBJECT</u></b> .....	<b>242</b>
<b><u>TRACE OBJECT</u></b> .....	<b>248</b>
<b><u>TRACES COLLECTION</u></b> .....	<b>250</b>
<b><u>TRANSFORM OBJECT</u></b> .....	<b>250</b>
<b><u>COM EXAMPLES</u></b> .....	<b>253</b>
<b><u>LEARNING ABOUT COM</u></b> .....	<b>268</b>
<b><u>SCPI COMMAND TREE</u></b> .....	<b>285</b>
<b><u>IEEE 488.2 COMMON COMMANDS</u></b> .....	<b>286</b>

<b><u>ABORT COMMAND</u></b> .....	<b>288</b>
<b><u>CALC:CORRECTION COMMANDS</u></b> .....	<b>289</b>
<b><u>CALC:DATA COMMANDS</u></b> .....	<b>290</b>
<b><u>CALC:FILTER COMMANDS</u></b> .....	<b>293</b>
<b><u>CALC:FORMAT COMMAND</u></b> .....	<b>297</b>
<b><u>CALC:FUNCTION COMMANDS</u></b> .....	<b>297</b>
<b><u>CALC:LIMIT COMMAND</u></b> .....	<b>300</b>
<b><u>CALC:MARKER COMMANDS</u></b> .....	<b>304</b>
<b><u>CALC:MATH COMMAND</u></b> .....	<b>314</b>
<b><u>CALC:NORMALIZE COMMANDS</u></b> .....	<b>315</b>
<b><u>CALC:PARAMETER COMMANDS</u></b> .....	<b>317</b>
<b><u>CALC:RDATA COMMAND</u></b> .....	<b>320</b>
<b><u>CALC:SMOOTHING COMMANDS</u></b> .....	<b>321</b>
<b><u>CALC:TRANSFORM COMMANDS</u></b> .....	<b>322</b>
<b><u>CONTROL COMMANDS</u></b> .....	<b>327</b>
<b><u>DISPLAY COMMANDS</u></b> .....	<b>338</b>
<b><u>FORMAT COMMANDS</u></b> .....	<b>346</b>
<b><u>HARDCOPY COMMAND</u></b> .....	<b>347</b>
<b><u>INITIATE COMMANDS</u></b> .....	<b>347</b>
<b><u>MEMORY COMMANDS</u></b> .....	<b>348</b>
<b><u>OUTPUT COMMAND</u></b> .....	<b>352</b>
<b><u>SENS:AVERAGE COMMANDS</u></b> .....	<b>352</b>
<b><u>SENSE:BANDWIDTH COMMAND</u></b> .....	<b>354</b>
<b><u>SENSE:CORRECTION COMMANDS</u></b> .....	<b>354</b>
<b><u>SENSE:CORRECTION:COLLECT:CKIT COMMANDS</u></b> .....	<b>363</b>

<b><u>SENSE:CORRECTION:CSET COMMANDS</u></b> .....	<b>373</b>
<b><u>SENSE:CORRECTION:COLLECT:GUIDED COMMANDS</u></b> .....	<b>375</b>
<b><u>SENSE:COUPLE COMMAND</u></b> .....	<b>380</b>
<b><u>SENSE:FREQUENCY COMMANDS</u></b> .....	<b>381</b>
<b><u>SENSE:POWER COMMAND</u></b> .....	<b>383</b>
<b><u>SENSE:ROSCILLATOR COMMAND</u></b> .....	<b>383</b>
<b><u>SENSE:SEGMENT COMMANDS</u></b> .....	<b>384</b>
<b><u>SENSE:SWEEP COMMANDS</u></b> .....	<b>390</b>
<b><u>SOURCE COMMANDS</u></b> .....	<b>394</b>
<b><u>SOURCE:POWER:CORRECTION COMMANDS</u></b> .....	<b>399</b>
<b><u>STATUS REGISTER COMMANDS</u></b> .....	<b>405</b>
<b><u>STATUS COMMAND KEYWORDS</u></b> .....	<b>413</b>
<b><u>SYSTEM COMMANDS</u></b> .....	<b>414</b>
<b><u>TRIGGER COMMANDS</u></b> .....	<b>416</b>
<b><u>SCPI EXAMPLES</u></b> .....	<b>418</b>
<b><u>SCPI EXAMPLE PROGRAMS</u></b> .....	<b>418</b>
<b><u>LEARNING ABOUT SCPI</u></b> .....	<b>443</b>
<b><u>REAR PANEL CONNECTORS</u></b> .....	<b>471</b>
<b><u>8753 COMMAND CROSS REFERENCE</u></b> .....	<b>488</b>

## 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 to the Analyzer. A word of warning: 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   Cal   Both)	MMEMory:STORe	
Recall States (Inst   Cal   Both)	MMEMory:LOAD	app.recall
<b>Manage Files</b>		
List Files	MMEMory:CATalog	
Copy Files	MMEMory:COPIY	
Move Files	MMEMory:MOVE	
Delete Files	MMEMory:DELEte	
<b>Manage Folders</b>		
Change	MMEMory:CDIRectory	
Delete	MMEMory:RDIRectory	
Make	MMEMory:MDIRectory	
<b>Print</b>		
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



## Channel Commands

Power | Average | 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
<b>Power Settings</b>		
Power ON   OFF	OUTP	app.SourcePowerState
Power Value	SOUR:POW1	chan.TestPortPower
Port Selection	SENS:SWE:SRCP	chan.TestPortPower
Couple Ports OFF   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	SOUR:POW:SLOP:STAT	app.PowerSlope

Auto		
Power Slope Value	SOUR:POW:SLOP	app.PowerSlope
Receiver Attenuation	SENS:POW:ATT	chan.ReceiverAttenuator
<b>Averaging</b>		
Average ONIOFF	SENS:AVER	chan.Average
Average Factor	SENS:AVER:COUN	chan.AveragingFactor
Return the Average Count		chan.AveragingCount
Average Restart	SENS:AVER:CLE	chan.AveragingRestart
<b>Manage Channels</b>		
Add		chans.Add
Make Active		app.ActiveChannel
Read Channel Number		chan.ChannelNumber
Read Number of Channels		chans.Count



## Sweep Commands

Power | Segment | 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   Pwr   CW   Seg)	SENS:SWE:TYPE	chan.SweepType
Sweep Generation (Stepped   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
<b>Power Sweep</b>		
Start Power	SOUR:POW:STAR	chan.StartPower
Stop Power	SOUR:POW:STOP	chan.StopPower
Center	SOUR:POW:CENT	
Span	SOUR:POW:SPAN	
<b>Segment Sweep</b>		
ONIOFF	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

<b>Trigger</b>		
Source (where trigger comes from)		
Trigger Source (Int   Ext   Manual)	TRIG:SOUR	app.TriggerSignal
Internal   Manual Trigger! (for Manual Source)	INIT:CONT INIT	app.ManualTrigger
Ext. Trigger Slope (Positive   Negative)	TRIG:LEV	app.TriggerSignal
<b>Scope (what is triggered)</b>		
Trigger Scope (Global   Channel)	TRIG:SCOP	app.TriggerType
<b>Channel Settings (how the channel responds to triggers)</b>		
Cont   Groups   Hold Continuous	SENS:SWE:MODE	chan.Continuous
Number of Groups	SENS:SWE:GRO:COUN	chan.NumberOfGroups
Hold Single		<u>chan.Hold</u> 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
<b>Perform an Unguided Calibration</b>		
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.CalculateErrorCoefficients
Display an Error Term	SENS:CORR:COEFFicient	
Isolation ON/OFF	SENS:CORR:ISOLation	cal.AcquireCalStandard
<b>Perform a Guided Cal</b>		
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	SENS:CORR:COLL:GUID:DESC	

Step	?	
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	
<b>Perform an ECAL</b>		
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		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
<b>Recall / Save / Apply a Calibration or Error Term</b>		
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	
<b>Cal Sets</b>		
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
<b>Correction Settings</b>		
CORR ONIOFF for a measurement	SENS:CORR	meas.ErrorCORR
Interpolation ONIOFF	SENS:CORR:INT	meas.InterpolateCORR
Extensions ONIOFF	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
<b>Modify Cal Kits</b>		
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
<b>Modify Cal Standards</b>		

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
<b>Power Calibration</b>		
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:ADD R	pwrCal.PowerMeterGPIBAddress
<b>Retrieve and Put Calibration Data</b>		
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
ONIOFF	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 Type (Normal   Fixed)	CALC:MARK:DISC CALC:MARK:TYPE	mark.Interpolated 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:THRESH	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		meas.Name

Name		
Read Measurement Number		meas.Number
Change Parameter		meas.ChangeParameter
Measurement Format	CALC:FORM	meas.Format
<b>Math</b>		
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 Multi Div)	CALC:MATH:FUNC	meas.TraceMath
<b>Smoothing</b>		
Smoothing ON/OFF	CALC:SMO	meas.Smoothing
Smoothing Aperture	CALC:SMO:APER	meas.SmoothingAperture
Smoothing Points	CALC:SMO:POIN	
<b>Statistics</b>		
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
<b>Limit Lines</b>		
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:START	limtseg.BeginStimulus
End Stimulus	CALC:LIM:SEGM:AMPL:STOP	limtseg.EndStimulus
Begin Response	CALC:LIM:SEGM1:AMPL:START	limtseg.BeginResponse
End Response	CALC:LIM:SEGM1:AMPL:STOP	limtseg.EndResponse
<b>Transform</b>		
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

<b>Gating</b>		
ONIOFF	CALC:FILT:TIME:STAT	gate.State
Type (BandPass, Notch)	CALC:FILT:TIME	gate.Type
Shape	CALC:FILT:GATE:TIME:SHA P	gat.Shape
Start	CALC:FILT:TIME:STAR	gate.Start
Stop	CALC:FILT:GATE:TIME:STO P	gate.Stop
Center	CALC:FILT:GATE:TIME:CEN T	gate.Center
Span	CALC:FILT:GATE:TIME:SPA N	gate.Span
<b>Window</b>		
Kaiser Beta	CALC:TRAN:TIME:KBES	trans.KaiserBeta
Impulse Width	CALC:TRAN:TIME:IMP:WIDT	trans.ImpulseWidth



## Scale Commands

<b>Description</b>	<b>SCPI</b>	<b>COM</b>
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

<b>Description</b>	<b>SCPI</b>	<b>COM</b>
Quit application		app.Quit
Preset	SYST:PRES	app.Preset
Reset		app.Reset
<b>Status Commands</b>		
Status Registers	GP-IB>Status	
*OPC;*WAI	GP-IB/Common_Commands	
<b>Events</b>		
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
<b>Macros</b>		
Execute Macro		app.ExecuteShortcut
Get Macro		app.GetShortcut
Delete Macro		app.DeleteShortCut
Put Macro		app.PutShortcut
<b>Rear Panel Connector Controls</b>		
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



## Data Commands

See a map of the data access locations

Description	SCPI	COM
<b>Get Measurement Data FROM the Analyzer</b>		
Get <b>complex</b> data from the specified location.		IArrayTrans.getComplex
Get typed <b>NAComplex</b> data from the specified location.		IArrayTrans.getNAComplex
Get <b>data pairs</b> from the specified location.		IArrayTrans.getPairedData
Get <b>scalar</b> data from the specified location.		IArrayTrans.getScalar
Get <b>variant</b> 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	
<b>Put Measurement Data INTO the Analyzer</b>		
Put <b>complex</b> data into the specified location.		IArrayTrans.putComplex
Put typed <b>NAComplex</b> data into the specified location.		IArrayTrans.putNAComplex
Put <b>scalar</b> data into the measurement result location.		IArrayTrans.putScalar
Put <b>complex Variant</b> data into the specified location.		IArrayTrans.putDataComplex
Put complex or formatted data into the measurement or memory result buffer	Calc:Data	
<b>Get Calibration Data FROM the Analyzer</b>		
Get <b>complex Error Term</b>		ICalData.GetErrorTermComple

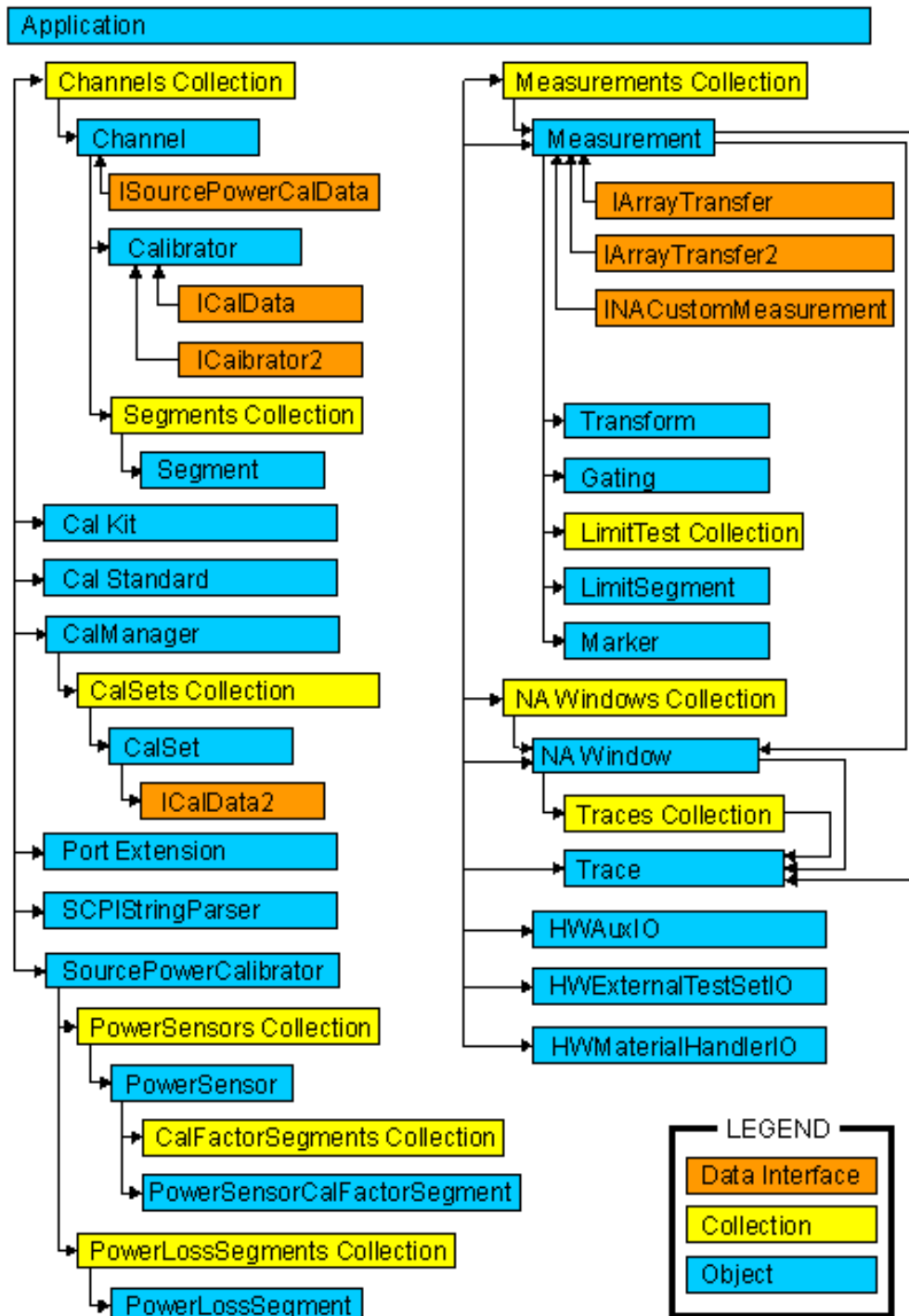
data		x
Get <b>variant Error Term</b> data	Calc:Data?	CalSet.getErrorTerm
Get <b>complex Standard</b> data		ICalData2.getStandardComple
		x
Get <b>variant Standard</b> data		CalSet.getStandard
<b>Put Calibration Data INTO the Analyzer</b>		
Put <b>complex Error Term</b> data		ICalData.putErrorTermComple
		x
Put <b>variant Error Term</b> data	Calc:Data	CalSet.putErrorTerm
Put <b>complex Standard</b> data		ICalData2.putStandardComple
		x
Put <b>variant Standard</b> data		CalSet.putStandard
<b>Get and Put Custom Measurement Data</b>		
Get and Put Custom data	Calc:Data	IArrayTransfer2 Interface



## PNA Object Model

---

See a list of obsolete commands.



## 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

## Application Object

### Application Object

#### 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.

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.
<b>Channel (object)</b>	
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
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

GetMaterialHandlerIO	interface Returns a handle to the MaterialHandlerIO interface
GetMaxChannels	Returns the maximum number of channels available on the PNA
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.
<b>Properties</b>	<b>Description</b>
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.
<b>Channels (collection)</b>	
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
<b>Measurements (collection)</b>	
<b>NAWindows (collection)</b>	

NumberOfPorts	Returns the number of hardware source ports on the PNA
Options	Returns the options on the analyzer
<b>PortExtension (object)</b>	
<b>SCPIStringParser (object)</b>	
<b>SourcePowerCalibrator (object)</b>	
SourcePowerState	Turns Source Power ON and OFF.
SystemImpedanceZ0	Sets the analyzer impedance value
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. <b>(Default property of this object)</b>
WindowState	Sets or returns the window setting of Maximized, Minimized, or Normal.
	Shared with the NAWindow Object
<b>Events</b>	<b>Description</b>
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  
ActivateWindow Method**

**About Windows**

<b>Description</b>	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 <b>Set</b> 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.
<b>VB Syntax</b>	<i>app</i> .ActivateWindow <i>n</i>
<b>Variable</b> <i>app</i> <i>n</i>	<b>(Type) - Description</b> An Application <b>(object)</b> <b>(long)</b> Number of the window to make active
<b>Return Type</b> <b>Default</b>	Window Object Not Applicable
<b>Examples</b>	<code>app.ActivateWindow 4</code>
<b>C++ Syntax</b>	<code>HRESULT ActivateWindow(long WindowNumber)</code>

**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**)

<i>event</i>	Event to monitor. Refer to list in Working with the Analyzer's Events
<i>state</i>	<b>(boolean)</b> <b>True</b> - monitor <b>False</b> - do not monitor
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.AllowEventMessage
<b>C++ Syntax</b>	HRESULT AllowEventMessage( tagNAEventID eventId, VARIANT_BOOL bAllow)
<b>Interface</b>	IApplication

**Write/Read** **About Analyzer Events**  
**AllowEventSeverity Method**

<b>Description</b>	Sets event filtering to monitor levels of severity.
<b>VB Syntax</b>	<i>app.AllowEventSeverity severity, state</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>severity</i>	<b>(enum naEventSeverity)</b> Choose from: naEventSeverityERROR naEventSeverityINFORMATIONAL naEventSeveritySUCCESS naEventSeverityWARNING
<i>state</i>	<b>(boolean)</b> <b>True</b> - monitor <b>False</b> - do not monitor
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.AllowEventSeverity
<b>C++ Syntax</b>	HRESULT AllowEventSeverity( tagNAEventSeverity severity, VARIANT_BOOL bAllow)
<b>Interface</b>	IApplication

**Write-only** **About Modifying Cal Kits**  
**BuildHybridKit Method**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.BuildHybridKit port1Kit, p1sex, port2Kit, p2sex, adapter, user kit</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>port1Kit</i>	<b>(enum NACalKit)</b> - Specifies the two kits to be used to build the hybrid kit. Choose from:
<i>port2Kit</i>	naCalKit_85032F_N50 naCalKit_85033E_3_5 naCalKit_85032B_N50



<i>p1sex</i>	naCalKit_85033D_3_5
<i>p2sex</i>	naCalKit_85038A_7_16
	naCalKit_85052C_3_5_TRL
	naCalKit_User7
	naCalKit_User8
	naCalKit_User9
	naCalKit_User10
	<b>(enum NAPortSex)</b> - Specifies the sex of the connector at that port.
	Choose from:
	<b>naMale</b>
	<b>naFemale</b>
	<b>naDon'tCare</b>
<i>adapter</i>	<b>(enum NAAadapter)</b> -Choose from:
	<b>naUserkit</b> - the electrical length of the adapter in the userKit specifications
	<b>naZeroLength</b> - no adapter
<i>userKit</i>	<b>(enum NACalKit)</b> - The Hybrid kit - Choose from the previous list of kits
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.BuildHybridKit naCalKit_85033E_3_5,naMale,naCalKit_85038A_7_16 ,naFemale,naUserkit,naCalKit_User8
<b>C++ Syntax</b>	HRESULT BuildHybridKit(tagNACalKit port1Kit, tagNAPortSex port1Sex, tagNACalKit port2Kit, tagNAPortSex port2Sex, tagNAAadapter adapter, tagNACalKit userKit)
<b>Interface</b>	IApplication

## Write-only CreateCustomMeasurement Method

## About Custom Measurements

<b>Description</b>	Creates a new custom measurement.
<b>VB Syntax</b>	<i>app.CreateCustomMeasurement</i> <i>chanNum,guid[,window]</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	<b>(object)</b> - An Application object
<i>chanNum</i>	<b>(long)</b> -Channel number used by the new measurement; can exist or be a new channel.
<i>guid</i>	<b>(string)</b> - 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.
<i>window</i>	<b>(long)</b> 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.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.CreateCustomMeasurement 1, "{12345678-56D3-11D5-AD50-00108334AE98}" "Not an actual custom measurement - for example purpose only
<b>C++ Syntax</b>	HRESULT CreateCustomMeasurement (long ChannelNum, BSTR guid,

**Interface** long windowNumber)  
IApplication

**Write-only**  
**CreateMeasurement Method**

**About Measurement Parameters**

---

**Description**  
**VB Syntax**

Creates a new measurement  
*app.CreateMeasurement chanNum,param,IPort[,window]*

---

**Variable**  
*app*  
*chanNum*

**(Type) - Description**  
Application (**object**)  
**(long)** - Channel number of the new measurement; can exist or be a new channel

*param*

**((string)** - New parameter. Choose from:

**S11 | S22 | S21 | S12**

Additionally, for 3-port analyzers only:

**S33 | S13 | S31 | S23 | S32**

For non-ratioed measurements:

**A | B | R1 | 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**

<i>I</i> Port	<b>(long)</b> - Load port if <i>param</i> is a reflection S-Parameter Ignored if <i>param</i> is a transmission S-Parameter Source port if <i>param</i> is anything other than an S-parameter
<i>window</i>	<b>(long)</b> Optional argument. Window number of the new measurement. Choose <b>1</b> to <b>4</b> . If unspecified, the measurement will be created in the Active Window.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.CreateMeasurement(1,"A/R1",1,0)
<b>C++ Syntax</b>	HRESULT CreateMeasurement(long ChannelNum, BSTR strParameter, long IPort, long windowNumber)
<b>Interface</b>	IApplication

**Write-only  
CreateS-Parameter Method - Obsolete**

**About Measurement Parameters**

<b>Description</b>	<b>Note:</b> 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.
<b>VB Syntax</b>	<i>app.CreateSParameter chan,recvr,source,[window]</i>
<b>Variable</b> <i>app</i> <i>chan</i> <i>recvr</i> <i>source</i> <i>window</i>	<b>(Type) - Description</b> Application ( <b>object</b> ) <b>(long integer)</b> - Channel number of the new measurement <b>(long integer)</b> - Port number of the receiver (1 or 2) <b>(long integer)</b> - Port number of the source (1 or 2) <b>(long integer)</b> - Optional argument. Window number of the new measurement. Choose <b>1</b> to <b>4</b> . If unspecified, the S-Parameter will be created in the Active Window.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT CreateSParameter(long ChannelNum, long RcvPort, long SrcPort, long windowNumber)
<b>Interface</b>	IApplication

**Write-only  
CreateSParameterEx Method**

**About Measurement Parameters**

<b>Description</b>	Creates a new S-Parameter measurement in an existing or new window and specifies the load port for 3-port devices.
<b>VB Syntax</b>	<i>app.CreateSParameter chan,recvr,source[,loadPort][,window]</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	Application ( <b>object</b> )
<i>chan</i>	<b>(long integer)</b> - Channel number of the new measurement
<i>recvr</i>	<b>(long integer)</b> - Port number of the receiver
<i>source</i>	<b>(long integer)</b> - Port number of the source
<i>loadPort</i>	<b>(long integer)</b> - Port number of the load. Required for reflection measurements of 3-port devices on 3-port PNAs.
<i>window</i>	<b>(long integer)</b> - Optional argument. Window number of the new measurement. Choose <b>1</b> to <b>4</b> . If unspecified, the S-Parameter will be created in the Active Window.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT CreateSParameter(long ChannelNum, long RcvPort, long SrcPort, long LoadPort, long windowNumber)
<b>Interface</b>	IApplication

**Write-only  
DeleteShortCut Method**

**About Macros**

<b>Description</b>	Removes a macro from the list of macros in the analyzer. Does not remove the file. <b>Note:</b> 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.
<b>VB Syntax</b>	<i>app.DeleteShortCut item</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>item</i>	<b>(long integer)</b> number of the macro to be deleted.
<b>Return Type</b>	Not Applicable

<b>Default</b>	Not Applicable
<b>Examples</b>	<i>app.DeleteShortCut 2</i>
<b>C++ Syntax Interface</b>	HRESULT DeleteShortcut(long Number ) IApplication

**Write/Read** **About Analyzer Events**  
**DisallowAllEvents Method**

---

<b>Description</b> <b>VB Syntax</b>	Sets event filtering to monitor NO eventst. <i>app.DisallowAllEvents</i>
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> An Application ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>app.DisallowAllEvents</i>
<b>C++ Syntax Interface</b>	HRESULT DisallowAllEvents() IApplication

**Write-only** **About Printing**  
**DoPrint Method**

---

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

**Write-only** **About Macros**  
**ExecuteShortcut Method**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.ExecuteShortcut index</i>
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application ( <b>object</b> )

<i>index</i>	<b>(long integer)</b> - Number of the macro stored in the analyzer.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	app.ExecuteShortcut 1
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT ExecuteShortcut(long index) IApplication

**Read-only  
GetAuxIO Method**

**About the AuxIO connector**

<b>Description VB Syntax</b>	This method returns the IAuxIO interface. <i>app</i> . <b>GetAuxIO</b>
<hr/>	
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> <b>(object)</b> - An Application object
<b>Return Type</b>	IHWAuxIO
<b>Default</b>	Not Applicable
<hr/>	
<b>Example</b>	Dim app As AgilentPNA835x.Application Dim aux As HWAuxIO Set aux = app.GetAuxIO
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT GetAuxIO (IHWAuxIO **pAux); IApplication

**Read-only  
GetCalManager Method**

**About Cal Sets**

<b>Description VB Syntax</b>	This method returns the ICalManager interface. <i>app</i> . <b>GetCalManager</b>
<hr/>	
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application <b>(object)</b>
<b>Return Type</b>	ICalManager*
<b>Default</b>	Not Applicable
<hr/>	
<b>Example</b>	dim app as AgilentPNA835x.Application dim mgr as CalManager set mgr = app.GetCalManager
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT GetCalManager( ICalManager **mgr); IApplication

**Read-only  
Get ExternalTestSetIO Method**

**About the External TestSet connector**

<b>Description VB Syntax</b>	This method returns the IExternalTestSetIO interface. <i>app</i> . <b>GetExternalTestSetIO</b>
----------------------------------	---

<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application ( <b>object</b> )
<b>Return Type</b>	IHWExternalTestSetIO
<b>Default</b>	Not Applicable
<b>Example</b>	Dim app As AgilentPNA835x.Application Dim ets As HWExternalTestSetIO Set ets = app.GetExternalTestSetIO
<b>C++ Syntax Interface</b>	HRESULT GetExternalTestSetIO (IHWExternalTestSetIO **ptestset); IApplication

**Read-only**  
**Get MaterialHandlerIO Method**

**About the MaterialHandler connector**

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

**Read-only**  
**GetMaxChannels Method**

<b>Description</b> <b>VB Syntax</b>	Returns the maximum number of channels available on the PNA. <i>value</i> = <i>app</i> . <b>GetMaxChannels</b>
<b>Variable</b> <i>value</i> <i>app</i>	<b>(Type) - Description</b> <b>(long integer)</b> - variable to store the returned value <b>(object)</b> - Application object
<b>Return Type</b>	Long Integer
<b>Default</b>	Not Applicable
<b>Example</b>	numChans = app.GetMaxChannels
<b>C++ Syntax Interface</b>	HRESULT GetMaxChannels( long* NumChans); IApplication

**Read-only  
GetShortcut Method**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app</i> .GetShortcut <i>index, title, path, arguments</i>
<b>Variable</b> <i>app</i> <i>index</i> <i>title</i> <i>path</i> <i>arguments</i>	<b>(Type) - Description</b> Application ( <b>object</b> ) <b>(long)</b> - Number of the macro. Use a number between <b>1</b> and <b>12</b> . <b>(string)</b> - <b>Title</b> of the specified macro. (Appears in the softkey label) <b>(string)</b> - <b>Pathname</b> of the specified macro. <b>(string)</b> - Arguments for the specified macro
<b>Return Type</b>	String
<b>Default</b>	Not Applicable
<b>Example</b>	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
<b>C++ Syntax</b>	HRESULT GetShortcut(long Number, BSTR* title, BSTR* pathname, BSTR* arguments)
<b>Interface</b>	IApplication
<b>Remarks</b>	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  
LaunchCalWizard Method**

**About the Cal Wizard**

---

<b>Description</b>	Launches the Cal Wizard on the PNA and does not return until the Cal Wizard is dismissed. <b>Note:</b> The Cal Wizard operates on the active measurement. Therefore, activate the measurement to be calibrated before launching the Cal Wizard.
<b>VB Syntax</b>	<i>success</i> = <i>app</i> .LaunchCalWizard ( <i>newCS</i> )
<b>Variable</b> <i>success</i>	<b>(Type) - Description</b> <b>(boolean)</b> - variable to store the returned value <b>True</b> - The Cal was completed <b>False</b> - The Cal was canceled without completing the calibration.
<i>app</i> <i>newCS</i>	<b>(object)</b> Application object <b>(boolean)</b>



	<p><b>True</b> - Cal will be performed on a new Cal Set.  <b>False</b> - 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.</p>
<b>Return Type</b>	Boolean
<b>Default</b>	Not Applicable
<hr/>	
<b>Example</b>	<pre>dim bSuccess as boolean dim bNewCalset as boolean bNewCalSet = false bSuccess = app.LaunchCalWizard( bNewCalSet)</pre>
<hr/>	
<b>C++ Syntax Interface</b>	<p><b>HRESULT</b> IApplication</p>

**Write-only  
ManualTrigger Method**

**About Triggering**

<b>Description</b>	Triggers the analyzer when <code>TriggerSignal = naTriggerManual</code> .
<b>VB Syntax</b>	<code>app.ManualTrigger [sync],[timeout]</code>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	Application ( <b>object</b> )
<i>sync</i>	<b>(boolean)</b> - Optional argument. A variable set to either True or False.
	<b>True</b> - The analyzer waits until the trigger is completed to process subsequent commands.
	<b>False</b> - Subsequent commands are processed immediately (the default setting)
<i>timeout</i>	<b>(long)</b> - Optional argument. If <i>sync</i> is true, <i>timeout</i> 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 <i>sync</i> is False, the timeout setting is ignored.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	<pre>' 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</pre>
<hr/>	
<b>C++ Syntax Interface</b>	<p><b>HRESULT</b> ManualTrigger(VARIANT_BOOL bSynchronize, long timeout) IApplication</p>

**Write/Read  
MessageText Method**

**About Analyzer Events**

<b>Description</b>	Returns text for the specified eventID
<b>VB Syntax</b>	<code>app.MessageText,eventID,message</code>

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

**Write-only  
Preset Method**

**Factory Preset Settings**

<b>Description</b>	<b>Application Object:</b> 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. <b>Channel Object:</b> Resets the channel (object) to factory defined default settings. Does NOT delete the current measurements or add a new measurement.
<b>VB Syntax</b>	<i>app.Preset</i> <i>chan.Preset</i>
<b>Variable</b> <i>app</i> <i>chan</i> <b>Return Type</b> <b>Default</b>	<b>(Type) - Description</b> An Application ( <b>object</b> ) A Channel ( <b>object</b> ) Not Applicable Not Applicable
<b>Examples</b>	app.Preset
<b>C++ Syntax Interface</b>	HRESULT Preset() IApplication IChannel

**Write-only  
PrintToFile Method**

**About Saving to File**

<b>Description</b> <b>VB Syntax</b>	Saves the screen image to a bitmap (.bmp) file. <i>app.PrintToFile filename</i>
<b>Variable</b> <i>app</i> <i>filename</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(string)</b> Name of the file to save the screen to. If you don't provide a

<b>Return Type</b>	suffix, the analyzer will append <b>.bmp</b> . The file is saved to the current working directory unless a valid full path name is specified.
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>app.PrintToFile "myfile" app.PrintToFile "c:\data\myfile.bmp"</pre>
<b>C++ Syntax Interface</b>	HRESULT PrintToFile(BSTR bstrFile) IApplication

## Write-only PutShortcut Method

## About Macros

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.PutShortcut index,title,path</i>
<b>Variable</b> <i>app</i> <i>index</i>	<b>(Type) - Description</b> Application ( <b>object</b> )
<i>title</i>	<b>(long)</b> - 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.
<i>path</i>	<b>(string)</b> - The name to be assigned to the macro
<b>Return Type</b>	<b>(string)</b> - Full pathname to the existing macro "executable" file.
<b>Default</b>	Not Applicable
<b>Examples</b>	app.PutShortcut 1,"Test","C:\Automation\MyTest.vbs"
<b>C++ Syntax Interface</b>	HRESULT PutShortcut(long Number, BSTR title, BSTR pathname) IApplication

## Write-only Quit Method

<b>Description</b>	Terminates the Network Analyzer application.
<b>VB Syntax</b>	<i>app.Quit</i>
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.Quit
<b>C++ Syntax Interface</b>	HRESULT Quit() IApplication
<b>Remarks</b>	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  
Recall Method**

**About Save/Recall**

---

<b>Description</b>	Recalls a measurement state, calibration state, or both from the hard drive into the analyzer.
<b>VB Syntax</b>	Use <code>app.Save</code> to save the measurement and calibration state. <code>app.Recall (filename.ext)</code>
<b>Variable</b> <i>app</i> <i>filename.ext</i>	<b>(Type) - Description</b> Application ( <b>object</b> ) <b>(string)</b> - Filename and extension of the file to be recalled.
<b>Return Type</b> <b>Default</b>	Extensions: <ul style="list-style-type: none"> <li>• .sta - Instrument State</li> <li>• .cal - Calibration file</li> <li>• .cst - Both Instrument State and Calibration file</li> </ul> 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 <i>filename.ext</i> argument. Not Applicable Not Applicable
<b>Examples</b>	<code>app.Recall (mystate.sta)</code> 'Recalls "mystate.sta" from the default folder <code>app.Recall ("C:\Program Files\Agilent\Network Analyzer\Documents\Newfolder\MyState.cst)</code> 'Recalls "mystate.cst" from the specified folder
<b>C++ Syntax Interface</b>	HRESULT Recall(BSTR bstrFile) IApplication

**Write-only  
Recall Kits Method**

**About Modifying Cal Kits**

---

<b>Description</b>	Recalls the calibration kits definitions that were stored with the <code>SaveKits</code> command.
<b>VB Syntax</b>	<code>app.RecallKits</code>
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	<code>app.RecallKits</code>

**C++ Syntax Interface**      HRESULT RecallKits()  
IApplication

**Write-only  
Reset Method**

**About Presetting the Analyzer**

---

**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 Interface**      HRESULT Reset()  
IApplication

**Write-only  
RestoreCalKitDefaults Method**

**About Modifying Cal Kits**

---

**Description**      Restores the original properties of the specified Cal Kit, overwriting 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 Interface**      HRESULT RestoreCalKitDefaults(tagNACalKit kit)  
IApplication

**Write-only  
RestoreCalKitDefaultsAll Method**

**About Modifying Cal Kits**

<b>Description</b>	Restores the original properties of ALL of the Cal Kits, overwriting the last definitions with the factory defaults.
<b>VB Syntax</b>	<i>app</i> .RestoreCalKitDefaultsAll
<b>Variable</b> <i>app</i>	<b>(Type) - Description</b> Application ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	app.RestoreCalKitDefaultsAll
<b>C++ Syntax</b>	HRESULT RestoreCalKitDefaultsAll()
<b>Interface</b>	IApplication

### Write-only Save Method

### About Save/Recall

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app</i> .Save( <i>filename.ext</i> )
<b>Variable</b> <i>app</i> <i>filename.ext</i>	<b>(Type) - Description</b> Application ( <b>object</b> ) <b>(string)</b> - Filename and extension of the file to be saved.
	Extensions:
	<ul style="list-style-type: none"> <li>• <b>.cst</b> - Saves both Instrument State and Cal Set reference - Recalls a calibrated measurement. (Recallable)</li> <li>• <b>.sta</b> - Saves Instrument State only - recalls the instrument state without calibration. (Recallable)</li> <li>• <b>.cal</b> - 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)</li> <li>• <b>.prn</b> - Saves a Bitmap of the screen (not recallable)</li> <li>• <b>.bmp</b> - Saves a Bitmap of the screen (not recallable)</li> <li>• <b>.s1p</b> - Saves 1-port measurement data (not recallable)</li> <li>• <b>.s2p</b> - Saves 2-port measurement data (not recallable)</li> </ul>
	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 <i>filename.ext</i> . argument.
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	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 Interface**

HRESULT Save(BSTR bstrFile)  
IApplication

---

**Write-only SaveKits Method**

**About Modifying Cal Kits**

---

**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**

*app*

**Return Type**

**Default**

**(Type) - Description**

Application (**object**)

Not Applicable

Not Applicable

---

**Examples**

app.SaveKits

---

**C++ Syntax Interface**

HRESULT SaveKits()  
IApplication

---

**Write/Read SetFailOnOverRange Method**

**About Analyzer Events**

---

**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**

*app*

*state*

**(Type) - Description**

An Application (**object**)

**(boolean)** -

**True (1)** - Overrange conditions report an error code

**False (0)** - Overrange conditions report a success code

Not Applicable

False (0)

---

**Return Type**

**Default**

---

**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 Interface**

HRESULT put\_SetFailOnOverRange(VARIANT\_BOOL mode)  
 IApplication

**Write-only  
 ShowStatusBar Method**
**About Display Formatting****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

**VB Syntax**

Remote or Local operation  
*app.ShowStatusBar state*

**Variable**

*app*  
*state*

**(Type) - Description**

Application (**object**)  
**(boolean)** -  
**True (1)** - Show the Status Bar  
**False (0)** - Hide the Status Bar

**Return Type Default**

Not Applicable  
 Not Applicable

**Examples**

*app.ShowStatusBar True*

**C++ Syntax Interface**

HRESULT ShowStatusBar (VARIANT\_BOOL bState)  
 IApplication

**Write-only  
 ShowStimulus Method**
**About Display Formatting****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**

*app*  
*state*

**(Type) - Description**

Application (**object**)  
**(boolean)** -  
**True (1)** - Show the Stimulus information  
**False (0)** - Hide the Stimulus information

**Return Type Default**

Not Applicable  
 Not Applicable

**Examples**

*app.ShowStimulus True*

**C++ Syntax**

HRESULT ShowStimulus(VARIANT\_BOOL bState)





**Read-only**  
**ActiveCalKit Property**

**About Calibration Kits**

---

<b>Description</b>	Returns a handle to the Active CalKit object. You can either <b>(1)</b> use the handle directly to access CalKit properties and methods, or <b>(2)</b> set a variable to the CalKit object. The variable retains a handle to the original object if another CalKit becomes active.
<b>VB Syntax</b>	1) <code>app.ActiveCalKit.&lt;setting&gt;</code> <b>or</b> 2) <code>Set cKit = app.ActiveCalKit</code>
<b>Variable</b> <i>app</i> <i>&lt;setting&gt;</i> <i>cKit</i>	<b>(Type) - Description</b> <b>(object)</b> - An Application object A CalKit property (or method) and arguments
<b>Return Type</b> <b>Default</b>	<b>(object)</b> - A CalKit object CalKit object None
<b>Examples</b>	Public cKit as calKit Set cKit = app.ActiveCalKit 'read
<b>C++ Syntax</b> <b>Interface</b>	HRESULT get_ActiveCalKit (ICalKit * kit) IApplication

**Read-only**  
**ActiveChannel Property**

**About Channels**

---

<b>Description</b>	Returns a handle to the Active Channel object. You can either <b>(1)</b> use the handle directly to access channel properties and methods, or <b>(2)</b> set a variable to the channel object. The variable retains a handle to the original channel if another channel becomes active.
<b>VB Syntax</b>	(1) <code>app.ActiveChannel.&lt;setting&gt;</code> <b>or</b> (2) <code>Set chan = app.ActiveChannel</code>
<b>Variable</b> <i>chan</i> <i>app</i> <i>&lt;setting&gt;</i>	<b>(Type) - Description</b> A Channel <b>(object)</b> An Application <b>(object)</b> A channel property (or method) and arguments
<b>Return Type</b> <b>Default</b>	Channel object Not applicable
<b>Examples</b>	1) <code>app.ActiveChannel.Averaging = 1</code> 2) Public chan as Channel Set chan = app.ActiveChannel
<b>C++ Syntax</b> <b>Interface</b>	HRESULT get_ActiveChannel( IChannel* *pVal) IApplication

**Read-only**

## ActiveMeasurement Property

---

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

## Read-only ActiveNAWindow Property

## About Windows

---

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

## Write/Read

## About Arrange Windows

## ArrangeWindows Property

---

<b>Description</b>	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 <code>app.WindowState</code> .
<b>VB Syntax</b>	<code>app.ArrangeWindows = value</code>
<b>Variable</b> <i>app</i> <i>value</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(enum NAWindowModes)</b> - Choose from: <b>0 - naTile</b> <b>1 - naCascade</b> <b>2 - naOverlay</b> <b>3 - naStack2</b> <b>4 - naSplit3</b> <b>5 - naQuad4</b>
<b>Return Type</b> <b>Default</b>	NAWindowModes naTile
<b>Examples</b>	<code>app.ArrangeWindow = naTile 'Write</code> <code>arrWin = app.ArrangeWindows 'Read</code>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT put_ArrangeWindows(tagNAWindowModes newVal) IApplication

## Write/Read CalKitType Property

## About Modifying Cal Kits

<b>Description</b>	Sets and returns a calibration kit type for calibration or to be used for kit modification. To get a handle to this kit, use <code>app.ActiveCalKit</code>
<b>VB Syntax</b>	<code>object.CalKitType = value</code>
<b>Variable</b> <i>object</i>  <i>value</i>	<b>(Type) - Description</b> A calkit ( <b>object</b> ) or An Application ( <b>object</b> ). <b>Note:</b> <code>app.CalKitType</code> and <code>calkit.calKitType</code> perform exactly the same function. <b>(enum naCalKit)</b> - Calibration Kit type. 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
<b>Return Type</b> <b>Default</b>	naCalKit Not Applicable

<b>Examples</b>	calkit.CalKitType = naCalKit_85038A_7_16 kitype = app.CalKitType
<b>C++ Syntax</b>	HRESULT get_CalKitType(tagNACalKit *pVal) HRESULT put_CalKitType(tagNACalKit newVal)
<b>Interface</b>	IApplication ICalKit

**Write/Read**  
**CoupledMarkers Property**

**About Coupled Markers**

<b>Description</b> <b>VB Syntax</b>	Sets and Reads the state of Coupled Markers (ON and OFF) <i>app.CoupledMarkers = state</i>
<b>Variable</b> <i>app</i> <i>state</i>	<b>(Type) - Description</b> <b>(object)</b> - An Application object <b>(boolean)</b> <b>False (0)</b> - Turns Coupled Markers OFF <b>True (1)</b> - Turns Coupled Markers ON
<b>Return Type</b>	Boolean <b>0</b> - OFF <b>1</b> - ON
<b>Default</b>	OFF (0)
<b>Examples</b>	app.CoupledMarkers = True 'Write coupl = app.CoupledMarkers 'Read
<b>C++ Syntax</b> <b>Interface</b>	IApplication

**Write/Read**  
**ExternalALC Property**

<b>Description</b> <b>VB Syntax</b>	Sets or returns the source of the analyzer leveling control. <i>app.ExternalALC = value</i>
<b>Variable</b> <i>app</i> <i>value</i>	<b>(Type) - Description</b> An Application <b>(object)</b> <b>(boolean)</b> - Choose from: <b>True (or 1)</b> - Leveling control supplied through the rear panel. <b>False (or 0)</b> - Leveling control supplied inside the analyzer
<b>Return Type</b> <b>Default</b>	Boolean 0
<b>Examples</b>	app.ExternalALC = True 'Write extALC = app.ExternalALC 'Read
<b>C++ Syntax</b>	HRESULT get_ExternalALC(VARIANT_BOOL *pVal) HRESULT put_ExternalALC(VARIANT_BOOL newVal)

**Interface** IApplication

### Write/Read GPIBMode Property

### About GPIB Fundamentals

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.GPIBMode value</i>
<b>Variable</b> <i>app value</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(enum NAGPIBMode)</b> -Choose either: <b>0</b> - naTalkerListener - the analyzer is a talker / listener <b>1</b> - naSystemController - the analyzer is the system controller
<b>Return Type</b> <b>Default</b>	Long Integer 0 - naTalkerListener
<b>Examples</b>	<code>app.GPIBMode = naTalkerListener 'Write mode = app.GPIBMode 'Read</code>
<b>C++ Syntax</b>	<code>HRESULT get_GPIBMode(tagGPIBModeEnum* eGpibMode) HRESULT put_GPIBMode(tagGPIBModeEnum eGpibMode)</code>
<b>Interface</b>	IApplication

### Read-only IDString Property

---

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

### Read-only NumberOfPorts Property

---

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

### Read-only Options Property

### About Options

---

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

### Write/Read SourcePowerState Property

### About Source Power

---

<b>Description</b> <b>VB Syntax</b>	Turns Source Power ON and OFF <i>app.SourcePowerState = state</i>
<b>Variable</b> <i>app</i> <i>state</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(boolean)</b> <b>False (0)</b> - Turns Source Power OFF <b>True (1)</b> - Turns Source Power ON
<b>Return Type</b>	Boolean <b>0</b> - Power OFF <b>1</b> - Power ON
<b>Default</b>	ON (1)
<b>Examples</b>	app.SourcePowerState = True 'Write pwr = app.SourcePowerState 'Read

<b>C++ Syntax</b>	HRESULT get_SourcePowerState(VARIANT_BOOL *pVal)
<b>Interface</b>	HRESULT put_SourcePowerState(VARIANT_BOOL newVal) IApplication

<b>Write/Read</b>	<b>About System Impedance</b>
<b>SystemImpedanceZ0 Property</b>	

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

<b>Write/Read</b>	<b>About Trigger Source</b>
<b>TriggerSignal Property</b>	

<b>Description</b>	Sets or returns the trigger source.
<b>VB Syntax</b>	<i>app.TriggerSignal = value</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>value</i>	<b>(enum NATriggerSignal) - Choose from:</b>
	<b>0 - naTriggerInternal</b> - free run
	<b>1 - naTriggerExternalPositive</b> - a trigger signal is generated when a TTL high is sensed on the external trigger pin of the Aux IO connector
	<b>2 - naTriggerExternalNegative</b> - a trigger signal is generated when a TTL low is sensed on the external trigger pin of the Aux IO connector.
	<b>3 - naTriggerManual</b> - manual trigger source; use <i>app.ManualTrigger</i> to send a trigger signal.
	<b>4 - naTriggerExternalHigh</b> - a trigger signal is generated when a TTL high is sensed on the external trigger pin of the Aux IO connector
	<b>5 - naTriggerExternalLow</b> - a trigger signal is generated when a TTL low is sensed on the external trigger pin of the Aux IO connector
<b>Return Type</b>	Long Integer
<b>Default</b>	naTriggerInternal



<b>Examples</b>	app.TriggerSignal = naTriggerExternalPositive 'Write trigsign = app.TriggerSignal 'Read
<b>C++ Syntax</b>	HRESULT get_TriggerSignal(tagNATriggerSignal *pSignal) HRESULT put_TriggerSignal(tagNATriggerSignal signal)
<b>Interface</b>	IApplication

**Write/Read.**  
**TriggerType Property**

**About Trigger**

<b>Description</b>	Sets or returns the trigger type which determines the scope of a trigger signal. <b>Note:</b> Point (Trigger Mode) is only available in Manual trigger and Channel (Trigger Type). If you change any channel to Global Trigger, Trigger Mode will be set to Measurement.
<b>VB Syntax</b>	<i>app.TriggerType = value</i>
<b>Variable</b> <i>app</i> <i>value</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(enum NATriggerType) - Trigger type. Choose from:</b> <b>0 - naGlobalTrigger</b> - a trigger signal is applied to all triggerable channels <b>1 - naChannelTrigger</b> - 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.
<b>Return Type Default</b>	Long Integer naGlobalTrigger
<b>Examples</b>	app.TriggerType = naGlobalTrigger 'Write trigtyp = app.TriggerType 'Read
<b>C++ Syntax</b>	HRESULT get_TriggerType(tagNATriggerType *pTrigger) HRESULT put_TriggerType(tagNATriggerType trigger)
<b>Interface</b>	IApplication

**Write/Read**  
**VelocityFactor Property**

**About Port Extensions**

<b>Description</b>	Sets the velocity factor to be used with Electrical Delay and Port Extensions.
<b>VB Syntax</b>	<i>app.VelocityFactor = value</i>
<b>Variable</b> <i>app</i> <i>value</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(double) - Velocity factor. Choose a number between: 0 and 10</b> (.66 polyethylene dielectric; .7 teflon dielectric) <b>Note:</b> to specify the electrical delay for reflection measurements (in both

<b>Return Type</b>	directions), double the velocity factor.
<b>Default</b>	Double 1
<b>Examples</b>	app.VelocityFactor = .66 'Write RelVel = app.VelocityFactor 'Read
<b>C++ Syntax</b>	HRESULT get_VelocityFactor(double *pVal) HRESULT put_VelocityFactor(double newVal)
<b>Interface</b>	IApplication

### Write/Read Visible Property

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.Visible = state</i>
<b>Variable</b> <i>app</i> <i>state</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) <b>(boolean)</b> <b>0</b> - Network Analyzer application <b>NOT</b> visible <b>1</b> - Network Analyzer application <b>IS</b> visible
<b>Return Type</b>	Boolean 0 - Not visible 1 - visible
<b>Default</b>	1
<b>Examples</b>	app.Visible = 0 'Write vis = app.Visible 'Read
<b>C++ Syntax</b>	HRESULT get_Visible(VARIANT_BOOL * bVisible) HRESULT put_Visible(VARIANT_BOOL bVisible)
<b>Interface</b>	IApplication

### About Analyzer Events

#### OnCalEvent

<b>Description</b>	Triggered by a calibration event. See a list of CAL Events.
<b>VB Syntax</b>	<b>Note:</b> Some Severe Events are also used as Error Messages Sub <i>app_OnCalEvent</i> (ByVal <i>eventID</i> As Variant, ByVal <i>chanNum</i> As Variant, ByVal <i>measNum</i> As Variant)
<b>Variable</b> <i>app</i> <i>eventID</i> <i>chanNum</i> <i>measNum</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) Code number of the event which occurred Channel Number of the event Measurement Number of the event

<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>Sub pna_OnCalEvent(ByVal eventID As Variant, ByVal channelNumber As Variant, ByVal measurementNumber As Variant) , MsgBox ("A Calibration event has occurred") End Sub</pre>
<b>C++ Syntax</b>	HRESULT OnCalEvent(VARIANT eventID, VARIANT channelNumber, VARIANT measurementNumber)
<b>Interface</b>	IApplication

### About Analyzer Events

#### OnChannelEvent

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

### About Analyzer Events

#### OnDisplayEvent

<b>Description</b>	Triggered by a display event. See a list of Display Events
<b>VB Syntax</b>	<p><b>Note:</b> Some Severe Events are also used as Error Messages</p> <pre>Sub <i>app_OnDisplayEvent</i>(ByVal <i>eventID</i> As Variant, ByVal <i>winNum</i> As Variant, ByVal <i>traceNum</i> As Variant)</pre>
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>eventID</i>	Code number of the event which occurred
<i>winNum</i>	Window Number of the event

<i>traceNum</i>	Trace Number of the event
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	Sub pna_OnDisplayEvent(ByVal eventID As Variant, ByVal windowNumber As Variant, ByVal traceNumber As Variant) MsgBox ("A Display event has occurred") End Sub
<hr/>	
<b>C++ Syntax</b>	HRESULT OnDisplayEvent(VARIANT eventID, VARIANT windowNumber, VARIANT traceNumber)
<b>Interface</b>	IApplication

### About Analyzer Events

#### OnHardwareEvent

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

### About Analyzer Events

#### OnMeasurementEvent

<b>Description</b>	Triggered by a measurement event. See a list of Measurement Events. <b>Note:</b> Some Severe Events are also used as Error Messages
<b>VB Syntax</b>	Sub <i>app_OnMeasurementEvent</i> (ByVal <i>eventID</i> As Variant, ByVal <i>measNum</i> As Variant)
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>eventID</i>	Code number of the event which occurred
<i>measNum</i>	Measurement Number of the event
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	Private Sub pna_OnMeasurementEvent(ByVal eventID As Variant, ByVal

```
measurementNumber As Variant)

MsgBox ("A Measurement event has occurred")

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** *app*  
*eventID*  
**Return Type** Not Applicable  
**Default** Not Applicable

---

**Examples** Private Sub pna\_OnSCPIEvent(ByVal eventID As Variant)  
MsgBox ("A SCPI event has occurred")  
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** *app*  
*eventID*  
*chanNum*  
**Return Type** Not Applicable  
**Default** Not Applicable

---

**Examples** Private Sub pna\_OnSystemEvent(ByVal eventID As Variant)  
MsgBox ("A System event has occurred")

End Sub

---

**C++ Syntax  
Interface**

HRESULT OnSystemEvent(VARIANT eventID)  
IApplication

### About Analyzer Events

#### OnUserEvent

---

**Description** Reserved for future use.  
**VB Syntax** Sub app\_OnUserEvent

### 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** *Object*  
**(Type) - Description**  
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

<i>n</i>	<p>equivalent:  Channels.Item(3).Averaging = 1  Channels(3).Averaging = 1</p> <p><b>(variant)</b> - number of the item in the collection.</p> <p><b>Note:</b> the Measurements and Traces collections allow you to specify the name of the measurement as a string. For example:  measCollection("CH_S11_1").InterpolateMarkers</p>
<b>Return Type</b>	<b>(Object)</b>
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	For i = 1 to Traces.Count -1 Traces.Item(i).YScale = .5dB Next i
<hr/>	
<b>C++ Syntax Interfaces</b>	HRESULT Item(VARIANT index, <interface>** pItem) ICalFactorSegments ICal Sets IChannels ILimitTest IMeasurements INaWindows IPowerLossSegments IPowerSensors ISegments ITraces
<b>Read-only Count Property</b>	
<hr/>	
<b>Description</b>	Returns the number of items in a collection of objects.
<b>VB Syntax</b>	<i>object</i> .Count
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>object</i>	Any of the following <b>(objects)</b> : Cal Sets collection CalFactorSegments collection Channels collection LimitTest collection Measurements collection NAWindows collection PowerLossSegments collection PowerSensors collection Segments collection Traces collection
<b>Return Type</b>	Long Integer
<b>Default</b>	Not applicable
<hr/>	
<b>Examples</b>	numofchans = chans.Count 'return the number of channels -Read
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT get_Count(long *p<interface>) ICal Sets ICalFactorSegments IChannels

ILimitTest  
 IMeasurements  
 INAWindows  
 IPowerLossSegments  
 IPowerSensors  
 ISegments  
 ITraces

**Read-only  
 Parent Property**

<b>Description</b>	Returns a handle to the parent object of the collection object being referred to in the statement. The parent property allows the user to traverse from an object back up the object hierarchy.
<b>VB Syntax</b>	<i>object</i> .Parent
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> Channels collection Channel object Measurements collection NAWindows collection Traces collection Segments collection PowerSensors collection CalFactorSegments collection PowerLossSegments collection
<b>Return Type</b> <b>Default</b>	Object Not Applicable
<b>Examples</b>	parentobj = chans.Parent 'returns a handle to the parent object (Application) of the chans collection. -Read
<b>C++ Syntax</b>	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
<b>Interface</b>	IChannels IChannel IMeasurements INAWindows ITraces ISegments IPowerSensors ICalFactorSegments



## Write-only Remove Method

<b>Description</b> <b>VB Syntax</b>	Removes an item from a collection of objects. <i>Object.Remove</i> <i>item</i>
<b>Variable</b> <i>Object</i>	<b>(Type) - Description</b> Any of the following ( <b>objects</b> ) CalFactorSegments collection Cal Sets collection Measurements collection NAWindows collection PowerLossSegments collection Segments collection <b>Note:</b> Segments, CalFactorSegments, and PowerLossSegments have an OPTIONAL argument [size] referring to the number of segments to remove, starting with the <i>item</i> parameter. <b>(variant)</b> - Item number to be removed
<i>item</i> <b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	Measurements.Remove 3 'Removes measurement 3 segments.Remove 2,20 'Removes 20 segments (2 - 21)
<b>C++ Syntax</b>	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
<b>Interface</b>	IMeasurements INAWindows ISegments ICalFactorSegments ICal Sets IPowerLossSegments

## Write/Read State Property

<b>Description</b> <b>VB Syntax</b>	Turns an Object ON and OFF. <i>object.State = value</i>
<b>Variable</b>	<b>(Type) - Description</b>

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

## Calibrator Object

## 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.SetCallInfo (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.CalculateErrorCoefficients
  - 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 cal (S11 and S22) and four response (thru) cal. 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.
CalculateErrorCoefficients	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 acquisition 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.
ECAL Isolation	Include Isolation in ECAL calibration
IsECALModuleFound	Tests communication between the PNA and ECAL Module
Simultaneous2PortAcquisition	Allows the use of 2 sets of standards at the same time.



**Write-only** **About ECAL Confidence Check**  
**AcquireCalConfidenceCheckECAL Method**

<b>Description</b>	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 <b>ECALCharacterization property</b> on the <b>ICalibrator2</b> interface. The default value of the ECALCharacterization property is <b>naECALFactoryCharacterization</b> .
<b>VB Syntax</b>	<code>cal.AcquireCalConfidenceCheckECAL Sparam[,ecalModule]</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>cal</i>	A Calibrator <b>(object)</b>
<i>Sparam</i>	<b>(String)</b> S-parameter to transfer confidence data to. This parameter must be present on the same channel as the calibrator object.
<i>ecalModule</i>	<b>(enum NAECALModule)</b> – Optional argument. ECal module. Choose from: 0 - naECALModule_A (default, if unspecified) 1 - naECALModule_B
<b>Return Type</b>	None
<b>Default</b>	Not applicable
<b>Examples</b>	<code>Cal.AcquireCalConfidenceCheckECAL "S11", naECALModule_A</code>
<b>C++ Syntax</b>	<code>HRESULT AcquireCalConfidenceCheckECAL( _bstr_t strParameter, enum NAECALModule ecalModule );</code>
<b>Interface</b>	ICalibrator

**Write-only** **About Calibration Standards**  
**AcquireCalStandard Method - Obsolete**

<b>Description</b>	<b>Note:</b> This command has been replaced by AcquireCalStandard2
--------------------	--

Method, which provides for acquisition of sliding load standards. All other functionality is identical.

*cal.AcquireCalStandard std[,index]*

## **VB Syntax**

---

### **Variable**

*cal*

*std*

### **(Type) - Description**

A Calibrator (**object**)

(**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

<b>Return Type</b>	is more than one standard required to cover the necessary frequency range. If unspecified, value is set to 0.
<b>Default</b>	None Not Applicable
<b>Examples</b>	Cal.AcquireCalStandard naSOLT_Thru "Write
<b>C++ Syntax</b>	HRESULT AcquireCalStandard(tagNACalClass enumClass, short standardNumber)
<b>Interface</b>	ICalibrator

## Write-only AcquireCalStandard2 Method

## About Calibration Standards

**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**  
*cal*  
*std*

**(Type) - Description**

A Calibrator (**object**)

(**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 - **naSlidelsSet** - slide is set for acquisition

2 - **naSlidelsDone** - this next acquisition will be the last. Calculations will then be performed.

**Return Type**  
**Default**

None  
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 CalculateErrorCoefficients Method

## About Performing a Calibration

**Description**

This method is the final call in a calibration process. It calculates error-correction terms and turns error-correction ON.

**VB Syntax**

*cal*.**CalculateErrorCoefficients**

**Variable**

**(Type) - Description**

<i>cal</i>	Calibrator ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	Cal.CalculateErrorCoefficients
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT CalculateErrorCoefficients() ICalibrator

**Write-only  
DoECAL1Port Method**

**About Calibration**

<b>Description</b>	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 <b>ECALCharacterization property</b> on the <b>ICalibrator2</b> interface. The default value of the ECALCharacterization property is <b>naECALFactoryCharacterization</b> .
<b>VB Syntax</b>	<code>cal.DoECAL1Port [port][,module]</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>cal</i>	A Calibrator ( <b>object</b> )
<i>port</i>	<b>(long integer)</b> Optional argument - Port number to calibrate. Choose from: 1 - Calibrate port 1 (default if unspecified) 2 - Calibrate port 2
<i>module</i>	<b>(enum NAECALModule)</b> Optional argument - ECAL module. Choose from:  0 - naECALModule_A - (default if unspecified) 1 - naECALModule_B
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	<code>cal.DoECAL1Port,2,naECALModule_B</code>
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT DoECAL1Port(long port, tagNAECALModule ecalModule); ICalibrator

**Write-only  
DoECAL2Port Method**

**About Calibration**

<b>Description</b>	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 <b>ECALCharacterization property</b> on the <b>ICalibrator2</b> interface. The default value of the ECALCharacterization property is <b>naECALFactoryCharacterization</b> .
<b>VB Syntax</b>	<code>cal.DoECAL2Port[portA][,portB][,module]</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>cal</i>	A Calibrator ( <b>object</b> )



<i>portA</i>	<b>(long integer)</b> 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)
<i>portB</i>	<b>(long integer)</b> 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)
<i>module</i>	<b>(enum NAECALModule)</b> – Optional argument. ECal module. Choose from: 0 - naECALModule_A (default, if unspecified) 1 - naECALModule_B
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>cal.DoECAL2Port,1,2,naECALModule_B</i>
<b>C++ Syntax</b>	HRESULT DoECAL2Port(long rcvport, long srcPort, tagNAECALModule ecalModule);
<b>Interface</b>	ICalibrator

**Write-only** **About ECAL Confidence Check**  
**DoneCalConfidenceCheckECAL Method**

---

<b>Description</b>	Concludes the Confidence Check and sets the ECal module back into the idle state.
<b>VB Syntax</b>	<i>cal.DoneCalConfidenceCheckECAL</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>cal</i>	A Calibrator <b>(object)</b>
<b>Return Type</b>	None
<b>Default</b>	None
<b>Examples</b>	<i>cal.DoneCalConfidenceCheckECAL</i>
<b>C++ Syntax</b>	HRESULT DoneCalConfidenceCheckECAL( );
<b>Interface</b>	ICalibrator

**Read-only**  
**GetECALModuleInfo Method**

---

<b>Description</b>	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**.

*moduleInfo = cal.GetECALModuleInfo (module)*

**VB Syntax**

**Variable**

*moduleInfo*  
*cal*  
*module*

**(Type) - Description**

**(string)** - variable to store the module information

A Calibrator **(object)**

**(enum NAECALModule)** – ECAL module. Choose from:

0 - naECALModule\_A

1 - naECALModule\_B

String

Not Applicable

**Return Type**

**Default**

**Examples**

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

**Example return string:**

ModelNumber: 85092-60007, SerialNumber: 01386, ConnectorType: N5FN5F RF2, Calibrated: 5 Jun 2000 , MinFreq: 50000000, MaxFreq: 9100000000

**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.

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

**VB Syntax**

**Variable**

*data*  
*cal*  
*term*

**(Type) - Description**

Variant array to store the data.

A Calibrator **(object)**

**(enum As NaErrorTerm)**. Choose from:

naErrorTerm\_Directivity\_Isolation

naErrorTerm\_Match

naErrorTerm\_Tracking

**(long integer)** - Receiver Port

**(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
Fwd Trans Tracking	naErrorTerm_Tracking	2	1

<b>Return Type</b>	Variant
<b>Default</b>	Not Applicable
<b>Examples</b>	Dim varError As Variant varError = cal.getErrorTerm(naErrorTerm_Tracking,2,1)
<b>C++ Syntax</b>	HRESULT getErrorTerm(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, VARIANT* pData)
<b>Interface</b>	ICalibrator

**Read-only**  
**GetStandard Method - Obsolete**

**About Cal Sets**

**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.

*data = cal.getStandard(class,rcv,src*

**VB Syntax**

**Variable**

*data*  
*cal*  
*class*

**(Type) - Description**

Variant array to store the data.

A Calibrator (**object**)

(**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*

*src*

**Return Type**

**Default**

---

**Examples**

---

**C++ Syntax**

**Interface**

---

**Write-only  
PutErrorTerm Method - Obsolete**

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

Dim varStd As Variant  
varStd = cal.getStandard(naSOLT\_Thru,2,1)

HRESULT raw\_getStandard(tagNACalClass  
stdclass, long ReceivePort, long SourcePort,  
VARIANT\* pData)  
ICalibrator

**About Measurement Calibration**

<b>Description</b>	<b>Note:</b> This command is replaced by CalSet.putErrorTerm.		
	Puts variant error term data into the error-correction buffer. See Accessing data.		
<b>VB Syntax</b>	Learn about reading and writing Calibration data. <i>cal.putErrorTerm(term,rcv, src, data)</i>		
<b>Variable</b>	<b>(Type) - Description</b>		
<i>cal</i>	A Calibrator <b>(object)</b>		
<i>term</i>	<b>(enum As NaErrorTerm)</b> naErrorTerm_Directivity_Isolation naErrorTerm_Match naErrorTerm_Tracking		
<i>rcv</i>	<b>(long integer)</b> - Receiver Port		
<i>src</i>	<b>(long integer)</b> - Source Port		
<i>data</i>	<b>(variant)</b> Error term data in a two-dimensional array (0:1, 0:numpts-1).		
<b>To get this Error Term</b>	<b>Specify these parameters:</b>		
	<i>term</i>	<i>rcv</i>	<i>src</i>
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
<b>Return Type</b>	Not Applicable		
<b>Default</b>	Not Applicable		
<b>Examples</b>	Dim varError As Variant varError = cal.putErrorTerm (naErrorTerm_Tracking,2,1,VarData)		
<b>C++ Syntax</b>	HRESULT putErrorTerm(tagNAErrorTerm ETerm, long ReceivePort, long SourcePort, VARIANT varData)		
<b>Interface</b>	ICalibrator		

---

Write-only

---

### PutStandard Method - Obsolete

---

<b>Description</b>	<b>Note:</b> 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

### Variable

*cal*  
*class*

*cal.putStandard class,rcv,src,data*

### (Type) - Description

A Calibrator (**object**)

(**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*  
*src*  
*data*

(**long**) - Receiver Port

(**long**) - Source Port

(**variant**) Two dimensional array (0:1, 0:points-1)

<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	Dim varStd (1,200) As Variant cal.putStandard naSOLT_Thru, 2, 1, varStd
<b>C++ Syntax</b>	HRESULT raw_putStandard(tagNACalClass stdclass, long ReceivePort, long SourcePort, VARIANT varData)
<b>Interface</b>	ICalibrator

## Write-only SaveCalSets Method

## About Cal Sets

<b>Description</b>	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: <ul style="list-style-type: none"> <li>• On Application Exit</li> <li>• When a Cal Set has been deleted</li> <li>• When a calibration has been performed through the front panel interface</li> <li>• When this method is called</li> </ul>
--------------------	---

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

<b>Variable</b>	<b>(Type) - Description</b>
<i>object</i>	<b>(object)</b> - A CalManager object or a Calibrator object
<b>Return Type</b>	None
<b>Default</b>	Not Applicable

**Example** calMgr.SaveCalSets

<b>C++ Syntax</b>	HRESULT SaveCalSets( );
<b>Interface</b>	ICalManager ICalibrator

## Write-only SetCallInfo Method

## About Performing a Calibration

<b>Description</b>	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.SetCallInfo(type,rcvPort,srcPort)*

<b>Variable</b>	<b>(Type) - Description</b>
<i>cal</i>	A Calibrator <b>(object)</b>

*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*  
*srcPort*  
**Return Type**  
**Default**

(long integer) - Receiver Port

(long integer) - Source Port

NACalType

7- **naCalType\_None**

---

**Examples**

`cal.setCallInfo(naCalType_Response_Open,1,1)`

---

**C++ Syntax**  
**Interface**

HRESULT SetCallInfo(tagNACalType calType, long portA, long portB)  
ICalibrator

## Read / Write AcquisitionDirection Property

## About Performing a Calibration

---

**Description**  
**VB Syntax**

Specifies the direction of each part of a 2-port calibration.  
`cal.AcquisitionDirection = value`

---

**Variable**  
*cal*  
*value*

**(Type) - Description**

A Calibrator (**object**)

(enum **NADirection**) - Choose from:

0 - **naForward** - measures the forward direction

1 - **naReverse** - measures the reverse direction

---

**Return Type**  
**Default**

Long Integer

naForward

---

**Examples**

`cal.AcquisitionDirection = naForward`

---

**C++ Syntax**  
**Interface**

HRESULT AcquisitionDirection(tagNADirection dir);  
ICalibrator

## Read/Write ECALIsolation Property

## About ECAL

---

**Description**

Specifies whether the acquisition of the ECal calibration should include isolation or not.



**VB Syntax** `cal.ECALIsolation=value`

**Variable**  
*cal*  
*value*

**(Type) - Description**  
A Calibrator **(object)**  
**(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 blIsolationState );`  
`VARIANT_BOOL GetECALIsolation( );`

**Interface** Calibrator

**Read only**  
**IsECALModuleFound Property**

**About ECAL**

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

**Variable**  
*moduleFound*

**(Type) - Description**  
**(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*  
*module*

**(object)** - A Calibrator object  
**(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

**Read / Write** **About Performing a Calibration**  
**Simultaneous2PortAcquisition Property**

---

<b>Description</b>	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).
<b>VB Syntax</b>	<i>cal.Simultaneous2PortAcquisition = state</i>
<b>Variable</b> <i>cal</i> <i>state</i>	<b>(Type) - Description</b> A Calibrator <b>(object)</b> <b>(boolean)</b> - Choose from: <b>True</b> - measures 2 ports simultaneously <b>False</b> - measures 1 port at a time
<b>Return Type</b> <b>Default</b>	Boolean False
<b>Examples</b>	<i>cal.Simultaneous2PortAcquisition = True</i>
<b>C++ Syntax</b>	HRESULT put_Simultaneous2PortAcquisition( VARIANT_BOOL bTwoSetsOfStandards) HRESULT Simultaneous2PortAcquisition( VARIANT_BOOL *bTwoSetsOfStandards)
<b>Interface</b>	ICalibrator

**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.

<b>Methods</b>	<b>Description</b>
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.
<b>Properties</b>	<b>Description</b>
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**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>segs.Add (item [ size])</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>segs</i>	<b>(object)</b> - A CalFactorSegments collection (object)
<i>item</i>	<b>(variant)</b> - Number of the new segment. If it already exists, a new segment is inserted at the requested position.
<i>size</i>	<b>(long integer)</b> - Optional argument. The number of segments to add, starting with item. If unspecified, value is set to 1.
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>segs.Add 1, 4</code> 'Adds segments 1,2,3 and 4
<b>C++ Syntax Interface</b>	HRESULT Add(VARIANT index, long size); ICalFactorSegments

**Cal Set Object**  
**CalSet Object**

---

**Description**

Use this interface to query and or change the contents of a Cal Set.  
 Learn about reading and writing Calibration data.

<b>Methods</b>	<b>Description</b>
CloseCalSet	Resets the CalType and port associations made in the OpenCal Set.
ComputeErrorTerms	Computes error terms for the CalType specified by a preceding OpenCal Set call.
Copy	Creates a new Cal Set and copies the current Cal Set data into it.
getErrorTerm	Retrieves variant error term data.
GetErrorTermList	Returns a list of error terms for the CalType specified by OpenCal Set
GetGUID	Returns the GUID identifying a Cal Set
getStandard	Retrieves variant data that was acquired for a specific cal standard.
GetStandardsList	Returns a list of standards required for CalType specified by OpenCal Set
HasCalType	Verifies that the Cal Set object contains the error terms required to apply the specified CalType to an appropriate measurement.
OpenCalSet	Opens the set and restricts access to a set of Error Terms.
putErrorTerm	Writes variant error term data into the error-correction buffer.
putStandard	Writes variant data that was acquired for a specific cal standard.

Save	Saves the current Cal Set to PNACalSets.dat.
StringToNACalClass	Converts string values from GetStandardsList into enumeration data
StringToNAErrorTerm2	Converts string values from GetErrorTermList into enumeration data

Properties	Description
Description	Descriptive string assigned to the Cal Set



**Write-only**  
**Close CalSet Method**

**About Cal Sets**

---

**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.

---

<b>VB Syntax</b>	<i>CalSet.CloseCalSet</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>CalSet</i>	<b>(object) - A Cal Set object</b>
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

---

**Examples** Cal Set.CloseCalSet

---

**C++ Syntax** HRESULT CloseCalSet  
**Interface** ICalSet

**Write-only**  
**ComputeErrorTerms Method**

**About Cal Sets**

---

**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*

<b>Variable</b> <i>CalSet</i>	<b>(Type) - Description</b> <b>(object)</b> - A Cal Set object
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	CalSet.ComputeErrorTerms
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT ComputeErrorTerms() ICalSet

## Write-only Copy Method

## About Cal Sets

<b>Description</b>	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.
<b>VB Syntax Variable</b> <i>CalSet</i>	<b>CalSet.Copy</b> <b>(Type) - Description</b> <b>(object)</b> - A Cal Set object
<b>Return Type</b>	Not Applicable
<b>Default</b>	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.

<b>C++ Syntax Interface</b>	HRESULT Copy( ICalSet** pCalSet); ICalSet
-----------------------------	--

<b>Description</b>	<p>Queries data from the Cal Set that was acquired for a specific standard. Learn more about reading and writing Cal Data using COM.</p> <p>Before calling this method you must open the Cal Set with OpenCalSet.. If the Cal Set is not open, this method returns E_NA_CalSet_ACCESS_DENIED.</p> <p>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.</p> <p><b>Note:</b> See also getErrorTermComplex on the ICalData2 interface to avoid using the variant data type.</p>
<b>VB Syntax</b>	<pre>data = CalSet.getErrorTerm setID, term, rcv, src</pre>
<b>Variable</b> <i>data</i> <i>CalSet</i> <i>setID</i>	<p><b>(Type) - Description</b>  Variant array to store the data.</p> <p>A Cal Set <b>(object)</b>  <b>(long integer)</b> – specifies which error term set to read data from. (0 is the master set of etterms.)</p> <p>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 &gt; 0) or the master array (setID = 0).</p> <p><b>Note:</b> Interpolated error terms are destroyed when no longer being used.</p> <p><b>(enum As NaErrorTerm2).</b> Choose from:</p> <ul style="list-style-type: none"> <li>0 - naET_Directivity (rcv = src)</li> <li>1 - naET_SourceMatch (rcv = src)</li> <li>2 - naET_ReflectionTracking (rcv = src)</li> <li>3 - naET_TransmissionTracking (rcv != src)</li> <li>4 - naET_LoadMatch (rcv != src)</li> <li>5 - naET_Isolation (rcv != src)</li> </ul> <p><b>(long integer)</b> - Receiver Port  <b>(long integer)</b> - Source Port</p>
<i>term</i>	
<i>rcv</i> <i>src</i>	
<b>Return Type</b> <b>Default</b>	Variant Not Applicable
<b>Examples</b>	<pre>Dim varError As Variant varError = CalSet.getErrorTerm(0,naET_TransmissionTracking,2,1)</pre>
<b>C++ Syntax</b>	<pre>HRESULT getErrorTerm(long setID, tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, VARIANT* pData)</pre>
<b>Interface</b>	ICalSet

**Write-only**  
**GetErrorTermList Method**

**About Cal Sets**

**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**  
**Variable**

*CalSet*

*SetID*

*count*

*strList*

**Return Type**

**Default**

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

**(Type) - Description**

**(object)** - A Cal Set object

(long) - specifies the error term set to query. Use 0 for the master set.

(long) - the number of error terms in the returned list

**(string)** - comma separated list of error terms found in Cal Set

Not Applicable

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**  
**GetGuid Method**

**About Cal Sets**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>value</i> = <i>CalSet</i> . <b>GetGuid</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(string)</b> - Variable to store the returned GUID
<i>CalSet</i>	<b>(object)</b> - A Cal Set object
<b>Return Type</b>	String
<b>Default</b>	Not Applicable
<b>Examples</b>	guid = CalSet.GetGuid 'Read
<b>C++ Syntax</b>	HRESULT GetGUID( BSTR* pGUIDString);
<b>Interface</b>	ICalSet

**Read-only  
GetStandard Method**

**About Cal Sets**

<b>Description</b>	<p>Queries data from the Cal Set that was acquired for a specific standard. Learn more about reading and writing Cal Data using COM.</p> <p>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.</p> <p>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.</p> <p><b>Note:</b> See also getStandardComplex on the ICalData2 interface to avoid using the variant data type.</p>
<b>VB Syntax</b>	<i>data</i> = <i>CalSet</i> . <b>getStandard</b> <i>standard</i> , <i>rcv</i> , <i>src</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>data</i>	Variant array to store the data.
<i>CalSet</i>	<b>A Cal Set (object)</b>
<i>standard</i>	<b>(enum NACalClass)</b> 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*

*src*

**Return Type**

**Default**

**(long integer)** - Receiver Port

**(long integer)** - Source Port

**(variant)** - two-dimensional array (0:1, 0:NumberOfPoints-1)

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**  
**GetStandardsList Method**

**About Cal Sets**

## 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

#### Variable

CalSet

count

list

#### Return Type

String

*CalSet.GetStandardsList (count, list)*

#### (Type) - Description

**(object)** - A Cal Set object

**(long [out])** - indicates the number of items returned in the list

**(string)** - Variable to store the returned Comma separated list of items.

String

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  
Interface**

HRESULT GetStandardsList( long\* count, BSTR\* list);  
ICalSet

**Read-only  
HasCalType Method**

**About Cal Sets**

**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**

naCalType\_Response\_Open  
naCalType\_Response\_Short  
\*naCalType\_Response\_Thru  
\*naCalType\_Response\_Thru\_And\_Isol  
naCalType\_OnePort  
naCalType\_TwoPort\_SOLT  
naCalType\_TwoPort\_TRL  
naCalType\_ThreePort\_SOLT

**Port arguments required**

Port1  
Port1  
Port1 (rcv), Port2 (src)  
Port1 (rcv), Port2 (src)  
Port1  
Port1, Port2  
Port1, Port2  
Port1, Port2, Port3

**VB Syntax  
Variable**  
*check*

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

**(Type) - Description**

**(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*  
*calType*

**(object)** - A Cal Set object

(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.

<i>port2</i>	<p>This specifies either:</p> <ul style="list-style-type: none"> <li>- the one significant port for an open/short response cal or a 1 port cal.</li> <li>- or one of the ports involved in a 2 or 3 port cal</li> <li>- or the <b>receive</b> port for a thru response / thru-isolation cal.</li> </ul> <p><b>(long)</b> - required for any caltype involving more than one port</p>
<i>port3</i>	<p>This specifies either:</p> <ul style="list-style-type: none"> <li>- one of the ports involved in a 2 or 3 port cal (order independent)</li> <li>- or the <b>source</b> port for a thru response / thru-isolation cal</li> </ul> <p><b>(long)</b> - required only for 3 port cal</p>
<b>Return Type</b>	<b>VARIANT_BOOL</b>
<b>Default</b>	Not Applicable
<b>Examples</b>	value = CalSet.HasCalType(naCalType_TwoPort_TRL, 1, 2)
<b>C++ Syntax</b>	HRESULT HasCalType( tagNACalType, long port1, long port2, long port3, BOOL *pVal);
<b>Interface</b>	ICalSet

**Read-only  
OpenCalSet Method**

**About Cal Sets**

**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.

<b>Caltype</b>	<b>Port arguments required</b>
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

---

<b>VB Syntax</b>	<i>CalSet.OpenCalSet</i> (CalType, port1, port2, port3)
<b>Variable</b>	<b>(Type) - Description</b>
<i>CalSet</i>	<b>(object) -</b> A Cal Set object
<i>CalType</i>	(enum as naCalType) - type of correction to be applied. Choose from
	<ul style="list-style-type: none"> <li>0 - naCalType_Response_Open</li> <li>1 - naCalType_Response_Short</li> <li>2 - naCalType_Response_Thru</li> <li>3 - naCalType_Response_Thru_And_Isol</li> <li>4 - naCalType_OnePort</li> <li>5 - naCalType_TwoPort_SOLT</li> <li>6 - naCalType_TwoPort_TRL</li> <li>7 - naCalType_None</li> <li>8 - naCalType_ThreePort_SOLT</li> </ul>
<i>port1</i>	<p><b>(long) -</b> required. This argument must be specified.</p> <p>This specifies either:</p> <ul style="list-style-type: none"> <li>- the one significant port for an open/short response cal or a 1 port cal.</li> <li>- or one of the ports involved in a 2 or 3 port cal</li> <li>- or the <b>receive</b> port for a thru response / thru-isolation cal.</li> </ul>
<i>port2</i>	<p><b>(long) -</b> required for any caltype involving more than one port</p> <p>This specifies either:</p> <ul style="list-style-type: none"> <li>- one of the ports involved in a 2 or 3 port cal (order independent)</li> <li>- or the <b>source</b> port for a thru response / thru-isolation cal</li> </ul>
<i>port3</i>	<p><b>(long) -</b> required only for 3 port cal</p> <p>This specifies either:</p> <ul style="list-style-type: none"> <li>- one of the ports involved in a 3 port cal (order independent)</li> </ul>
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>CalSet.OpenCalSet</i> naCalType_ThreePort_SOLT, 3,2,1
<b>C++ Syntax</b>	HRESULT OpenCalSet ( naCalType, port1, [optional] port2, [optional] port3);
<b>Interface</b>	ICalSet

**Write-only**  
**PutErrorTerm Method**

**About Cal Sets**

---

<b>Description</b>	<p>Puts error term data into the Cal Set. Learn more about reading and writing Cal data using COM</p> <p>Before calling this method you must open the Cal Set with OpenCal Set..</p>
--------------------	--

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.

*CalSet.putErrorTerm (term, rcv, src, data)*

## VB Syntax

### Variable

*CalSet*

*term*

*rcv*

*src*

*data*

### Return Type

Default

### Examples

#### (Type) - Description

**(object)** - A Cal Set object

**(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)

**(long integer)** - Receiver Port

**(long integer)** - Source Port

**(variant)** Error term data in a two-dimensional array (0:1, 0:numpts-1).

Not Applicable

Not Applicable

```
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****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.

*obj.putStandard class, rcv, src, data*

---

**VB Syntax****Variable**

*obj*

*class*

**(Type) - Description**

**(object)** - A Calibrator or Cal Set object  
**(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*  
*src*  
*data*

**Return Type**

**Default**

---

**Examples**

**(long)** - Receiver Port

**(long)** - Source Port

**(variant)** Two dimensional array ( NUMPTS, 2)

Not Applicable

Not Applicable

```
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 Save Method

### About Cal Sets

#### Description

Saves the current Cal Set to the PNA Cal Sets.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

#### Variable

*CalSet*

#### Return Type

#### Default

*CalSet.Save*

#### (Type) - Description

**(object)** - A Cal Set object

Not Applicable

Not Applicable

#### Examples

```
myCalSet.Save
```

See Copy Method for an example application of this command.

#### C++ Syntax

#### Interface

```
HRESULT Save();
```

```
ICalSet
```

### Read-only StringToNACalClass Method

### About Cal Sets

#### 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

#### Variable

*CalSet*

*list*

*std*

*CalSet.StringToNACalClass* (*list, std, rcv, src*)

#### (Type) - Description

**(object)** - A Cal Set object

**(string)** - a string containing the textual description of the standard.

**(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*  
*src*  
**Return Type**  
**Default**

**(long)** - port number of the receiver  
**(long)** - port number of the source  
 Not Applicable  
 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**  
**StringToNAErrorTerm2 Method**

**About Cal Sets**

---

<b>Description</b>	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
<b>VB Syntax</b>	<i>Cal Set</i> . <b>StringToNAErrorTerm2</b> ( <i>list, eterm, rcv, src</i> )
<b>Variable</b>	<b>(Type) - Description</b>
<i>Cal Set</i>	<b>(object)</b> - A Cal Set object
<i>list</i>	<b>(string)</b> - a string containing the textual description of the error term.
<i>eterm</i>	<b>(enum As NAErrorTerm2)</b> . 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)
<i>rcv</i>	<b>(long)</b> - port number of the receiver
<i>src</i>	<b>(long)</b> - port number of the source
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	CalSet.StringToNAErrorTerm2 str, term, rcv, src
<b>C++ Syntax</b>	HRESULT StringToNAErrorTerm2 (BSTR* str, NAErrorTerm2* item, long *rcv, long *src);
<b>Interface</b>	ICalSet

**Write / Read**  
**Description Property**

**About Cal Sets**

---

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

## 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

---

#### 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 calKitType and, in turn, assigns the "Active cal kit" 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_85032B_N50
Set ck1 = app.ActiveCalKit

app.CalKitType = naCalKit_85038A_7_16
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: 7-16 Model 85038
ck2: 7-16 Model 85038
```

Method	Description
getCalStandard	Returns a handle to a calibration standard for modifying its definitions.
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	Maps a standard device to a cal class.



## Write-only GetCalStandard Method

## About Modifying Cal Kits

<b>Description</b>	Returns a handle to a calibration standard for modifying its definitions. To select a standard for performing a calibration (use <code>Calibrator.AcquireCalStandard</code> ).
<b>VB Syntax</b>	<code>calkit.GetCalStandard(index)</code>
<b>Variable</b> <i>calkit</i> <i>index</i>	<b>(Type) - Description</b> A calKit ( <b>object</b> ) <b>(long)</b> - Number of calibration standard. Choose <b>1</b> to <b>8</b> ; (there are 8 cal standards in every kit).
<b>Return Type</b> <b>Default</b>	calStandard Not Applicable
<b>Examples</b>	Dim short As CalStandard Set short = calKit.getCalStandard(1) short.label = "myShort"
<b>C++ Syntax</b>	HRESULT GetCalStandard(long standardNumber, ICalStandard **pCalStd)
<b>Interface</b>	ICalKit

## Write/Read Name (CalKit) Property

## About Modifying Cal Kits

<b>Description</b> <b>VB Syntax</b>	Sets and Returns a name for the selected calibration kit. <code>calKit.Name = value</code>
<b>Variable</b>	<b>(Type) - Description</b>

<i>calKit</i> <i>value</i>	A CalKit ( <b>object</b> ). <b>(string)</b> -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).
<b>Return Type</b>	String
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>calKit.Name = "MyCalKit" 'Write</code> <code>KitName = calKit.Name 'Read</code>
<b>C++ Syntax</b>	HRESULT get_Name(BSTR *pVal) HRESULT put_Name(BSTR newVal)
<b>Interface</b>	ICalKit

### Write/Read PortLabel Property

### About Modifying Cal Kits

<b>Description</b>	Sets and returns the label on the calibration kit Port for the calibration wizard.
<b>VB Syntax</b>	<code>calKit.Portlabel (portNum) = value</code>
<b>Variable</b> <i>calKit</i> <i>portNum</i> <i>value</i>	<b>(Type) - Description</b> A CalKit ( <b>object</b> ) <b>(long integer)</b> - number of the port to be labeled. Choose either <b>1</b> or <b>2</b> <b>(string)</b> - Label that is visible in the calibration wizard.
<b>Return Type</b>	String
<b>Default</b>	Depends on the Cal Kit.
<b>Examples</b>	<code>calKit.PortLabel = "MyCalKit" 'Write</code> <code>kitLabel = calKit.PortLabel 'Read</code>
<b>C++ Syntax</b>	HRESULT get_PortLabel(long port, BSTR *pVal) HRESULT put_PortLabel(long port, BSTR newVal)
<b>Interface</b>	ICalKit

### Write/Read StandardForClass Property

### About Modifying Cal Kits

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

<i>&lt;value&gt;</i>	<b>Class</b>	<b>Description</b>
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**

---

**Description**

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

Methods	Description
CreateCalSet	Creates a new Cal Set
DeleteCalSet	Deletes a Cal Set
GetCalSetByGUID	Get a handle to a Cal Set
GetCalSetCatalog	Gets a list of Cal Sets
GetCalSetUsageInfo	Returns the Cal Set ID and Error Term ID currently in use
SaveCalSets	Writes new or changed Cal Sets to disk Shared with the Calibrator Object
Properties	
<b>CalSets (collection)</b>	

**Write-only  
CreateCalSet Method**

---

**About Cal Sets**

**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.



<b>VB Syntax</b>	<i>calMgr.CreateCalSet (chan)</i>
<b>Variable</b> <i>calMgr</i> <i>chan</i>	<b>(Type) - Description</b> <b>(object)</b> - A CalManager object <b>(long)</b> - channel number of the new Cal Set.
<b>Return Type</b> <b>Default</b>	ICal Set Interface Not Applicable
<b>Example</b>	<i>calMgr.CreateCalSet 1</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT CreateCalSet( long ChannelNumber, ICal Set** pCal Set); ICalManager

### Write-only DeleteCalSet Method

### About Cal Sets

**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)*

<b>Variable</b> <i>calMgr</i> <i>GUID</i>	<b>(Type) - Description</b> <b>(object)</b> - A CalManager object <b>(string)</b> - GUID number of the Cal Set to be deleted
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable

### Example

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

strGUID = cs.GetGUID
calMgr.DeleteCalSet strGUID
```

<b>C++ Syntax</b> <b>Interface</b>	HRESULT DeleteCalSet( BSTR strGUID); ICalManager
---------------------------------------	---

### Read-only Get CalSetByGUID Method

### About Cal Sets

**Description** Requests a Cal Set by GUID. Returns an ICal Set interface.

**VB Syntax** *calMgr.GetCalSetByGUID (GUID)*

<b>Variable</b>	<b>(Type) - Description</b>
-----------------	-----------------------------

<i>calMgr</i>	<b>(object)</b> - A CalManager object
<i>GUID</i>	<b>(string)</b> - GUID of the Cal Set being requested.
<b>Return Type</b>	Interface object
<b>Default</b>	Not Applicable
<hr/>	
<b>Example</b>	calMgr.GetCalSetByGUID (2B893E7A-971A-11d5-8D6C-00108334AE96)
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT GetCalSetByGUID( BSTR* strGUID, ICal Set* pCalSet); ICalManager

**Read-only  
GetCalSetCatalog Method**

**About Cal Sets**

<b>Description</b>	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}
<b>VB Syntax</b>	<i>value</i> = <i>calMgr</i> . <b>GetCalSetCatalog</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(string)</b> - Variable to store the returned GUID list
<i>calMgr</i>	<b>(object)</b> - A CalManager object
<b>Return Type</b>	String
<b>Default</b>	Not Applicable
<hr/>	
<b>Example</b>	<i>value</i> = <i>calMgr</i> .GetCalSetCatalog
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT GetCalSetCatalog( BSTR); ICalManager

**Read-only  
GetCalSetUsageInfo Method**

**About Cal Sets**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>calMgr</i> . <b>GetCalSetUsageInfo</b> ( <i>chan</i> , <i>GUID</i> , <i>EtermID</i> )

<b>Variable</b>	<b>(Type) - Description</b>
<i>calMgr</i>	<b>(object)</b> - A CalManager object
<i>chan</i>	<b>(long [in])</b> - channel of the Cal Set being requested
<i>GUID</i>	<b>(string [out])</b> - 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.
<i>EtermID</i>	<b>(long [out])</b> - variable to store the error term ID being requested. If the returned argument is greater than 0, the set is being interpolated.
<b>Return Type</b>	String , Long Integer
<b>Default</b>	Not Applicable
<b>Example</b>	calMgr.GetCalSetUsagelInfo (1, GUID, EtermID)
<b>C++ Syntax</b>	HRESULT GetCalSetUsagelInfo (long IChannel, BSTR* CalSetGUID, long* etermSetID);
<b>Interface</b>	ICalManager

## CalStandard Object

### CalStandard Object

#### 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.

Method	
None	
Property	Description
C0	Sets and Returns the C0 (C-zero) value (the first capacitance value) for the calibration standard, when the Type is set to "naOpen".
C1	Sets and Returns the C1 value (the second capacitance value) for the calibration standard, when the Type is set to "naOpen".
C2	Sets and Returns the C2 value (the third capacitance value) for the calibration standard, when the Type is set to "naOpen".
C3	Sets and Returns the C3 value (the fourth capacitance value) for the calibration standard, when the Type is set to "naOpen".
Delay	Sets and Returns the electrical delay value for the calibration standard.
L0	Sets and Returns the L0 (L-zero) value (the first inductance value) for the calibration standard, when the Type is set to "naShort".
L1	Sets and Returns the L1 value (the second inductance value) for the calibration standard, when the Type is set to "naShort"..
L2	Sets and Returns the L2 value (the third inductance value) for the calibration standard, when the Type is set to "naShort"..
L3	Sets and Returns the L3 value (the third inductance value) for the calibration standard, when the Type is set to "naShort"..
Label	Sets and Returns the label for the calibration standard.
loss	Sets and Returns the insertion loss for the calibration standard.
Maximum Frequency	Sets and Returns the maximum frequency for the calibration standard.
Medium	Sets and Returns the media type of the calibration standard.
Minimum Frequency	Sets and Returns the minumum frequency for the calibration standard.
Type	Sets and Returns the type of calibration standard. Selections are: <b>naOpen, naShort,</b>

Z0	<b>naLoad, naThru, naArbitraryImpedance and naSliding.</b> Sets and Returns the characteristic impedance for the calibration standard.
TZReal	Sets and Returns the TZReal value (the Real Terminal Impedance value) for the calibration standard, when the Type is set to "naArbitraryImpedance".
TZImag	Sets and Returns the TZImag value (the Imaginary Terminal Impedance value) for the calibration standard, when the Type is set to "naArbitraryImpedance".



### Write/Read C0 Property

### About Modifying Cal Kits

<b>Description</b>	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 <i>calstd.C0 = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for C0 in picofarads
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>calstd.C0 = 15</code> 'Write the value of C0 to 15picofarads <code>cap0 = calstd.C0</code> 'Read the value of C0
<b>C++ Syntax</b>	HRESULT get_C0(float *pVal) HRESULT put_C0(float newVal)
<b>Interface</b>	ICalStandard

### Write/Read C1 Property

### About Modifying Cal Kits

<b>Description</b>	Sets and Returns the C1 value (the second capacitance value) for the calibration standard. To set the other capacitance values, use C0, C2, C3 <i>calstd.C1 = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for C1 in picofarads
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>calstd.C1 = 15</code> 'Write the value of C1 to 15picofarads <code>cap1 = calstd.C1</code> 'Read the value of C1

**C++ Syntax** HRESULT get\_C1(float \*pVal)  
 HRESULT put\_C1(float newVal)  
**Interface** ICalStandard

**Write/Read  
 C2 Property**

**About Modifying Cal Kits**

---

**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  
*calstd.C2 = value*

**VB Syntax**

---

**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  
 C3 Property**

**About Modifying Cal Kits**

---

**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  
*calstd.C3 = value*

**VB Syntax**

---

**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  
Delay Property**

**About Modifying Cal Kits**

---

<b>Description</b>	Sets and Returns the electrical delay value for the calibration standard.
<b>VB Syntax</b>	<i>calstd.Delay = value</i>
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Electrical delay in seconds
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.Delay = .00015</i> 'Write the Delay .00015 seconds <i>stdDelay = calstd.Delay</i> 'Read the value of Delay
<b>C++ Syntax</b>	HRESULT get_Delay(float *pVal) HRESULT put_Delay(float newVal)
<b>Interface</b>	ICalStandard

**Write/Read  
L1 Property**

**About Modifying Cal Kits**

---

<b>Description</b>	Sets and Returns the L1 value (the second inductance value) for the calibration standard.
<b>VB Syntax</b>	To set the other inductance values, use L0, L2, L3 <i>calstd.L1 = value</i>
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for L1 in picohenries
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.L1 = 15</i> 'Write the value of L1 = 15picohenries <i>Induct1 = calstd.L1</i> 'Read the value of L1
<b>C++ Syntax</b>	HRESULT get_L1(float *pVal) HRESULT put_L1(float newVal)
<b>Interface</b>	ICalStandard

**Write/Read  
L2 Property**

**About Modifying Cal Kits**

---

<b>Description</b>	Sets and Returns the L2 value (the third inductance value) for the
--------------------	--

	calibration standard. To set the other inductance values, use L0, L1, L3 <i>calstd.L2 = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for L2 in picohenries
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.L2 = 15</i> 'Write the value of L2 to 15picohenries <i>Induct2 = calstd.L2</i> 'Read the value of L2
<b>C++ Syntax</b>	HRESULT get_L2(float *pVal) HRESULT put_L2(float newVal)
<b>Interface</b>	ICalStandard

### Write/Read L3 Property

### About Modifying Cal Kits

<b>Description</b>	Sets and Returns the L3 value (the third inductance value) for the calibration standard. To set the other inductance values, use L0, L1, L2 <i>calstd.L3 = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for L3 in picohenries
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.L3 = 15</i> 'Write the value of L3 to 15picohenries <i>Induct3 = calstd.L3</i> 'Read the value of L3
<b>C++ Syntax</b>	HRESULT get_L3(float *pVal) HRESULT put_L3(float newVal)
<b>Interface</b>	ICalStandard

### Write/Read L0 Property

### About Modifying Cal Kits

<b>Description</b>	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 <i>calstd.L0 = value</i>
<b>VB Syntax</b>	
<b>Variable</b>	<b>(Type) - Description</b>

<i>calstd</i>	A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for L0 in picohenries
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	calstd.L0 = 15 'Write the value of L0 = 15picohenries Induct0 = calstd.L0 'Read the value of L0
<hr/>	
<b>C++ Syntax</b>	HRESULT get_L0(float *pVal) HRESULT put_L0(float newVal)
<b>Interface</b>	ICalStandard

### Write/Read Label Property

### About Modifying Cal Kits

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

### Write/Read Loss Property

### About Modifying Cal Kits

<b>Description</b>	Sets and Returns the insertion loss for the calibration standard.
<b>VB Syntax</b>	<i>calstd.loss</i> = <i>value</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>calstd</i>	A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Insertion loss in Mohms / sec. (MegaOhms per second of electrical delay)
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	calstd.loss = 3.5e9 'Write



stdLoss = calstd.loss 'Read the value of Loss

---

<b>C++ Syntax</b>	HRESULT get_Loss(float *pVal)
<b>Interface</b>	HRESULT put_Loss(float newVal) ICalStandard

---

<b>Write/Read</b>	<b>About Modifying Cal Kits</b>
<b>MaximumFrequency Property</b>	

---

<b>Description</b>	Sets and Returns the maximum frequency for the calibration standard.
<b>VB Syntax</b>	<i>calstd.MaximumFrequency = value</i>

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>calstd</i>	A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(double)</b> - Maximum frequency in Hertz.
<b>Return Type</b>	Double
<b>Default</b>	Not Applicable

---

<b>Examples</b>	calstd.MaximumFrequency = 9e9 'Write maxFrequency = calstd.MaximumFrequency 'Read
-----------------	--

---

<b>C++ Syntax</b>	HRESULT get_MaximumFrequency(double *pVal)
<b>Interface</b>	HRESULT put_MaximumFrequency(double newVal) ICalStandard

---

<b>Write/Read</b>	<b>About Modifying Cal Kits</b>
<b>Medium Property</b>	

---

<b>Description</b>	Sets and Returns the media type of the calibration standard.
<b>VB Syntax</b>	<i>calstd.Medium = value</i>

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>calstd</i>	A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(enum NACalStandardMedium)</b> - Medium of the transmission line of the standard. Choose from: <b>0 - naCoax</b> - Coaxial Cable <b>1 - naWaveGuide</b>
<b>Return Type</b>	Long Integer
<b>Default</b>	Not Applicable

---

<b>Examples</b>	calstd.Medium = naCoax 'Write stdMedium = calstd.Medium 'Read
-----------------	--

---

<b>C++ Syntax</b>	HRESULT get_Medium(tagNACalStandardMedium *pVal)
<b>Interface</b>	HRESULT put_Medium(tagNACalStandardMedium newVal) ICalStandard

Write/Read  
MinimumFrequency Property

About Modifying Cal Kits

---

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

Write/Read.  
Type (calstd) Property

About Modifying Cal Kits

---

<b>Description</b> <b>VB Syntax</b>	Sets and Returns the type of calibration standard. <i>calstd.Type = value</i>
<b>Variable</b> <i>calstd</i>  <i>value</i>  <b>Return Type</b> <b>Default</b>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard. <b>(enum NACalStandardType)</b> -Choose from: <b>0 - naOpen</b> <b>1 - naShort</b> <b>2 - naLoad</b> <b>3 - naThru</b> Long Integer Not Applicable
<b>Examples</b>	<i>calstd.Type = naOpen 'Write</i> <i>standardtype = calstd.Type 'Read</i>
<b>C++ Syntax</b>	HRESULT get_Type(tagNACalStandardType *pVal) HRESULT put_Type(tagNACalStandardType newVal)
<b>Interface</b>	ICalStandard

Write/Read  
TZImag Property

About Modifying Cal Kits

<b>Description</b>	Sets and Returns the TZImag value (the Imaginary Terminal Impedance value) for the calibration standard. Only applicable when "Type" is set to <b>naArbitraryImpedance</b> .
<b>VB Syntax</b>	To set the other resistance values, use TZReal <i>calstd.TZImag = value</i>
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for TZImag in Ohms
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.TZImag = 15</i> 'Write the value of TZImag to 15 Ohms <i>imp0 = calstd.TZImag</i> 'Read the value of TZImag
<b>C++ Syntax</b>	HRESULT get_TZImag(float *pVal) HRESULT put_TZImag(float newVal)
<b>Interface</b>	ICalStandard2

**Write/Read  
TZReal Property**

**About Modifying Cal Kits**

<b>Description</b>	Sets and Returns the TZReal value (the real Terminal Impedance value) for the calibration standard. Only applicable when "Type" is set to <b>naArbitraryImpedance</b> .
<b>VB Syntax</b>	To set the other resistance values, use TZImag <i>calstd.TZReal = value</i>
<b>Variable</b> <i>calstd</i>	<b>(Type) - Description</b> A CalStandard ( <b>object</b> ). Use calKit.GetCalStandard to get a handle to the standard.
<i>value</i>	<b>(single)</b> - Value for TZReal in Ohms
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>calstd.TZReal = 15</i> 'Write the value of TZReal to 15 Ohms <i>imp0 = calstd.TZReal</i> 'Read the value of TZReal
<b>C++ Syntax</b>	HRESULT get_TZReal(float *pVal) HRESULT put_TZReal(float newVal)
<b>Interface</b>	ICalStandard2

**Write/Read  
Z0 Property**

**About Modifying Cal Kits**

<b>Description</b>	Sets and Returns the characteristic impedance for the calibration standard.
<b>VB Syntax</b>	<i>calstd.Z0 = value</i>

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

## Channel Object

### Channel Object

#### 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.

Method	Description
Abort	Aborts the current measurement sweep on the channel.
AveragingRestart	Clears and restarts averaging of the measurement data.
Continuous	The channel continuously responds to trigger signals.
getSourcePowerCalData	Returns requested source power calibration data, if it exists.
GetXAxisValues	Returns the channel's X-axis values into a dimensioned Variant array.
GetXAxisValues2	Returns the channel's X-axis values into a dimensioned NON-Variant array.

Hold	Puts the Channel in Hold - not sweeping.
Next_IFBandwidth	A function that returns the Next higher IF Bandwidth value.
NumberOfGroups	Sets the Number of trigger signals the channel will receive.
Preset	Resets the channel to factory defined settings.
PreviousIFBandwidth	Returns the previous IF Bandwidth value.
putSourcePowerCalData	Inputs source power calibration data to this channel for a specific source port.
SelectCal Set	Specifies the Cal Set to use for the Channel
Single	Channel responds to one trigger signal from any source (internal, external, or manual). Then channel switches to Hold.
Property	Description
AlternateSweep	Sets sweeps to either alternate or chopped.
Attenuator	Sets or returns the value of the attenuator control for the specified port number.
AttenuatorMode	Sets or returns the mode of operation of the attenuator control for the specified port number.
Averaging	Turns trace averaging ON or OFF for all measurements on the channel.
AveragingCount	Returns the number of sweeps that have been averaged into the measurements.
AveragingFactor	Specifies the number of measurement sweeps to combine for an average.
<b>Calibrator (object)</b>	
centerFrequency	Sets or returns the center frequency of the channel. Shared with the Segment Object
channelNumber	Returns the Channel number. Shared with the Measurement Object
CouplePorts	Turns ON and OFF port power coupling.
CWFrequency	Set the Continuous Wave (CW) frequency.
DwellTime	Sets or returns the dwell time for the channel. Shared with the Segment Object
FrequencySpan	Sets or returns the frequency span of the channel. Shared with the Segment Object
IFBandwidth	Sets or returns the IF Bandwidth of the channel. Shared with the Segment Object
NumberOfPoints	Sets or returns the Number of Points of the channel. Shared with the Segment Object
Parent	Returns a handle to the parent object of the channel.
PowerSlope	Sets or returns the Power Slope value.
ReceiverAttenuator	Sets or returns the value of the specified receiver attenuator control.
Segments (collection)	
SourcePowerCorrection	Turns source power correction ON or OFF for a specific source port.
StartFrequency	Sets or returns the start frequency of the channel. Shared with the Segment Object
StartPower	Sets the start power of the analyzer when sweep type is set to Power Sweep.
StopFrequency	Sets or returns the stop frequency of the channel. Shared with the Segment Object
StopPower	Sets the Stop Power of the analyzer when sweep type is set to Power Sweep.
SweepGenerationMode	Sets the method used to generate a sweep: continuous ramp (analog) or discrete steps (stepped).

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



### Write-only Abort Method

### About Triggering

---

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

### Write-only AveragingRestart Method

### About Averaging

---

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

**Write-only  
Continuous Method**

**About Triggering**

---

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

**Read-only  
getSourcePowerCalData Method**

**About Source Power Cal**

---

<b>Description</b>	Retrieves (as variant data type) requested source power calibration data, if it exists, from this channel. <b>Note:</b> This method returns a variant which is less efficient than methods available on the ISourcePowerCalData interface
<b>VB Syntax</b>	<i>data</i> = <i>chan</i> . <b>getSourcePowerCalData</b> <i>sourcePort</i>
<b>Variable</b> <i>data</i> <i>chan</i> <i>sourcePort</i>	<b>(Type) - Description</b> <b>(variant)</b> – Array to store the data. <b>(object)</b> – A Channel object <b>(long integer)</b> – The source port for which calibration data is being requested.
<b>Return Type</b> <b>Default</b>	Variant array – automatically dimensioned to the size of the data. Not Applicable
<b>Examples</b>	Dim varData As Variant Const port1 As Long = 1 varData = <i>chan</i> .getSourcePowerCalData port1 'Print the data For i = 0 to <i>chan</i> .NumberOfPoints - 1 Print varData(i) Next i
<b>C++ Syntax Interface</b>	HRESULT getSourcePowerCalData(long sourcePort, VARIANT *pData); IChannel

**Write-only  
GetX-axisValues2 Method**

**About Segment Sweep**

---

<b>Description</b>	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.

chan.GetXAxisValues2 numPts,data

### VB Syntax

---

#### Variable

*chan*

*numPts*

*data*

#### Return Type

Default

#### (Type) - Description

**(object)** - A Channel object

**(long integer)** - Number of data points in the channel

**(double)** single dimensioned array of data matching the number of points in the channel.

Not applicable

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 Interface

```
HRESULT GetXAxisValues2(long* pNumValues, double* stimulus)
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.

*data* = *chan*.GetXAxisValues

### VB Syntax

---

#### Variable

*data*

*chan*

#### Return Type

Default

#### (Type) - Description

Variant array to store the data.

A Channel **(object)**

Variant

Not Applicable



**Examples**

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

**C++ Syntax Interface**

```
HRESULT GetXAxisValues (VARIANT* xData)
IChannel
```

**Write-only  
Hold Method**

**About Triggering**

**Description** Puts the Channel in Hold - not sweeping.  
**VB Syntax** *chan.Hold [sync]*

**Variable**  
*chan*  
*[sync]*

**(Type) - Description**  
A Channel (**object**)  
**(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 Interface**

```
HRESULT Hold(VARIANT_BOOL bWait)
IChannel
```

**Write-only  
NextIFBandwidth Method**

**About Dynamic Range**

**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**  
*chan*  
*bw*

**(Type) - Description**  
A Channel (**object**)  
**(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
```

---

<b>C++ Syntax Interface</b>	HRESULT Next_IFBandwidth (double *pVal) IChannel
-----------------------------	---

---

<b>Write-only NumberOfGroups Method</b>	<b>About Triggering</b>
---	-------------------------

---

<b>Description</b>	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.
--------------------	---

<b>VB Syntax</b>	To begin sweeping the number of groups, send app.Continuous <i>chan.NumberOfGroups num, sync</i>
------------------	---

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

<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
--------------------------------------	----------------------------------

<b>Examples</b>	chan.NumberOfGroups
-----------------	---------------------

---

<b>C++ Syntax Interface</b>	HRESULT NumberOfGroups(long count, VARIANT_BOOL bWait) IChannel
-----------------------------	--

---

<b>Write-only PreviousIFBandwidth Method</b>	<b>About Dynamic Range</b>
--	----------------------------

---

<b>Description</b>	A function that returns the previous IF Bandwidth value. Use to retrieve the list of available IFBandwidth settings.
--------------------	--

<b>VB Syntax</b>	<i>chan.Previous_IFBandwidth bw</i>
------------------	-------------------------------------

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

<b>Examples</b>	<b>Public pnbw As Double 'declare variable outside of procedure</b> 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
-----------------	--

---

<b>C++ Syntax</b>	HRESULT Previous_IFBandwidth (double *pVal)
-------------------	---

**Interface** IChannel

**Write-only  
putSourcePowerCalData Method**

**About Source Power Cal**

---

<b>Description</b>	Inputs source power calibration data (as variant data type) to this channel for a specific source port.
<b>VB Syntax</b>	<i>chan</i> . <b>getSourcePowerCalData</b> <i>sourcePort</i> , <i>data</i>
<b>Variable</b> <i>chan</i> <i>sourcePort</i>  <i>data</i>	<b>(Type) - Description</b> <b>(object)</b> – A Channel object <b>(long integer)</b> – The source port for which calibration data is being requested. <b>(variant)</b> – Array of source power cal data being input.
<b>Return Type</b> <b>Default</b>	None Not Applicable
<b>Examples</b>	chan.putSourcePowerCalData 1, varData
<b>C++ Syntax</b> <b>Interface</b>	HRESULT putSourcePowerCalData(long sourcePort, VARIANT varData); IChannel

**Write-only  
SelectCalSet Method**

---

<b>Description</b>	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. <b>Note:</b> 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
<b>VB Syntax</b>	<i>channel</i> . <b>SelectCalSet</b> <i>GUID</i> , <i>restore</i>
<b>Variable</b> <i>channel</i> <i>GUID</i> <i>restore</i>	<b>(Type) - Description</b> <b>(object)</b> - A Channel object <b>(string)</b> - GUID number of the Cal Set to select <b>(boolean)</b> - <b>True (1)</b> - The stimulus stored with the cal set will be applied to the channel. <b>False (0)</b> - 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
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable

<b>Example</b>	channel.SelectCalSet GUID, 1
<b>C++ Syntax Interface</b>	HRESULT SelectCalSet (BSTR strGUID, bool bRestore); IChannel

### Write-only Single Method

### About Triggering

<b>Description</b>	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).
<b>VB Syntax</b>	<i>chan</i> .Single [ <i>sync</i> ]
<b>Variable</b> <i>chan</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> )
[ <i>sync</i> ]	<b>(boolean)</b> -Optional argument. A variable set to either True or False. <b>True</b> - The analyzer waits until the trigger is completed to process subsequent commands. <b>False</b> - Subsequent commands are processed immediately.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	sync = True chan.Single sync
<b>C++ Syntax Interface</b>	HRESULT Single(VARIANT_BOOL bWait) IChannel

### Write/Read AlternateSweep Property

### About Sweeping

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

**Interface** IChannel

**Read-only  
Application Property**

---

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

**Write/Read  
AttenuatorMode Property**

**About Attenuation**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>chan.AttenuatorMode(portNum) = value</i>
<b>Variable</b> <i>chan</i> <i>portNum</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(long)</b> - Port number ( <b>1</b> or <b>2</b> ) of attenuator control to be changed. <b>(enum NAModes)</b> - Choose from: <b>0 - naAuto</b> - Attenuator control set to automatic. The analyzer will set the attenuator control appropriately to deliver the specified power at the source. <b>1 - naManual</b> - Specify the attenuator setting using chan.Attenuator (which automatically sets AttenuatorMode = naManual).
<b>Return Type</b> <b>Default</b>	NAModes 0 - Auto
<b>Examples</b>	chan.AttenuatorMode(1) = naAuto 'Write attn = chan.AttenuatorMode(1) 'Read
<b>C++ Syntax</b>	HRESULT get_AttenuatorMode(long port, tagNAModes* pVal) HRESULT put_AttenuatorMode(long port, tagNAModes newVal)

**Interface** IChannel

**Write/Read**  
**Attenuator Property**

**About Attenuation**

---

<b>Description</b>	Sets or returns the value of the attenuator control for the specified port number. Sending this command automatically sets AttenuatorMode to Manual.
<b>VB Syntax</b>	<i>chan</i> .Attenuator( <i>portNum</i> ) = <i>value</i>
<b>Variable</b> <i>chan</i> <i>portNum</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(long integer)</b> - Port number ( <b>1</b> or <b>2</b> ) of attenuator control to be changed. <b>(double)</b> - Attenuator value in dB in 10dB steps. Choose any Long Integer between <b>0</b> and <b>70</b> 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.
<b>Return Type</b> <b>Default</b>	Double 20 dB
<b>Examples</b>	chan.Attenuator(1) = 20 'Write attn = chan.Attenuator(cnum) 'Read
<b>C++ Syntax</b>	HRESULT get_Attenuator(long port, double *pVal) HRESULT put_Attenuator(long port, double newVal)
<b>Interface</b>	IChannel

**Write/Read**  
**Averaging Property**

**About Averaging**

---

<b>Description</b>	Turns trace averaging ON or OFF for all measurements on the channel. Averaging is only allowed on ratioed measurements; not on single input measurements.
<b>VB Syntax</b>	<i>chan</i> .Averaging = <i>state</i>
<b>Variable</b> <i>chan</i> <i>state</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(boolean)</b> <b>0</b> - Turns averaging OFF <b>1</b> - Turns averaging ON
<b>Return Type</b> <b>Default</b>	Boolean 0
<b>Examples</b>	chan.Average = 1 'Write averg = chan.Averaging 'Read
<b>C++ Syntax</b>	HRESULT get_Averaging(BOOL *pVal)

**Interface** HRESULT put\_Averaging(BOOL newVal)  
IChannel

**Read-only**  
**AveragingCount Property**

**About Averaging**

---

**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** *chan*  
*value*  
**Return Type** Long Integer  
**Default** Not Applicable

**(Type) - Description**  
A Channel (**object**)  
**(Long Integer)** - Variable to store the returned count

---

**Example** avgcount = chan.AveragingCount

---

**C++ Syntax** HRESULT get\_AveragingCount(long\* count)  
**Interface** IChannel

**Write/Read**  
**AveragingFactor Property**

**About Averaging**

---

**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** *chan*  
*value*  
**Return Type** Long Integer  
**Default** 1

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

---

**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**  
**CenterFrequency Property**

**About Frequency**

---

<b>Description</b>	Sets or returns the center frequency of the channel <b>or</b> Sets or returns the center frequency of the segment.
<b>VB Syntax</b>	<i>object</i> . <b>centerFrequency</b> = <i>value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>or</b> A Segment ( <b>object</b> )
<i>value</i>	<b>(double)</b> - Center frequency in Hertz. Choose any number between the <b>minimum</b> and <b>maximum</b> frequencies of the analyzer.
<b>Return Type</b> <b>Default</b>	Double Center of the frequency range
<b>Examples</b>	chan.centerFrequency = 4.5e9 'sets the center frequency of a linear sweep for the channel object -Write centfreq = chan.centerFrequency 'Read
<b>C++ Syntax</b>	HRESULT get_CenterFrequency(double *pVal) HRESULT put_CenterFrequency(double newVal)
<b>Interface</b>	IChannel ISegment

**Read-only**  
**ChannelNumber Property**

**About Channels**

---

<b>Description</b> <b>VB Syntax</b>	Returns the Channel number of the Channel or Measurement object. <i>object</i> . <b>ChannelNumber</b>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>or</b> A Measurement ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Long Integer Not applicable
<b>Examples</b>	chanNum = chan.ChannelNumber 'returns the channel number chanNum = meas.ChannelNumber 'returns the channel number of the measurement
<b>C++ Syntax</b> <b>Interface</b>	HRESULT get_ChannelNumber(long *pVal) IChannel IMeasurement



**Write/Read**  
**CouplePorts Property**

**About Power Coupling**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>chan.CouplePorts = value</i>
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(enum NAStates)</b> Choose from: <b>0 - NaOff</b> - Turns coupling OFF <b>1 - NaOn</b> - Turns coupling ON
<b>Return Type</b>	Long Integer <b>1</b> - ON <b>0</b> - OFF
<b>Default</b>	NaON (1)
<b>Examples</b>	chan.CouplePorts = NaOff 'Write couplport = chan.CouplePorts 'Read
<b>C++ Syntax</b>	HRESULT get_CouplePorts(tagNAStates *pState) HRESULT put_CouplePorts(tagNAStates newState)
<b>Interface</b>	IChannel

**Write/Read**  
**CW Frequency Property**

**About CW Frequency**

---

<b>Description</b>	Set the Continuous Wave (CW) frequency. Must first send chan.SweepType = naCWTimeSweep
<b>VB Syntax</b>	<i>chan.CWFrequency = value</i>
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(double)</b> CW frequency. Choose any number between: the <b>minimum</b> and <b>maximum</b> frequency limits of the analyzer Units are Hz
<b>Return Type</b>	Double
<b>Default</b>	1e9
<b>Examples</b>	chan.CWFrequency = 5e9 'Write cwfreq = chan.CWFrequency 'Read
<b>C++ Syntax</b>	HRESULT put_CWFrequency(double newVal) HRESULT get_CWFrequency(double *pVal)
<b>Interface</b>	IChannel

**Write/Read**  
**DwellTime Property**

**About Dwell Time**

---

<b>Description</b>	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 = <b>naSteppedSweep</b> (not naAnalogSweep).  Sets or returns the dwell time of a specified sweep segment. <i>object.DwellTime = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) or A Segment ( <b>object</b> )
<i>value</i>	<b>(double)</b> - Dwell Time in seconds. Choose any number between: <b>0</b> and <b>100e-3</b>
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	<pre>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</pre>
<b>C++ Syntax</b>	HRESULT get_DwellTime(double *pVal) HRESULT put_DwellTime(double newVal)
<b>Interface</b>	IChannel ISegment

**Write/Read**  
**FrequencySpan Property**

**About Frequency Range**

---

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

**Write/Read**  
**IFBandwidth Property**

**About IF Bandwidth**

---

<b>Description</b>	Sets or returns the IF Bandwidth of the channel. Sets or returns the IF Bandwidth of the segment.
<b>VB Syntax</b>	<i>object</i> .IFBandwidth = <i>value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) or A Segment ( <b>object</b> )
<i>value</i>	<b>(double)</b> - IF Bandwidth in Hz. Choose from: <b>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  </b> 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.)
<b>Return Type</b>	Double
<b>Default</b>	3500
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_IFBandwidth(double *pVal); HRESULT put_IFBandwidth(double newVal);
<b>Interface</b>	IChannel ISegment

---

**Write/Read**  
**NumberOfPoints Property**

**About Number of Points**

---

<b>Description</b>	Sets or returns the Number of Points of the channel. Sets or returns the Number of Points of the segment.
<b>VB Syntax</b>	<i>object</i> .NumberOfPoints = <i>value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) or A Segment ( <b>object</b> )
<i>value</i>	<b>(long)</b> - Number of Points. For channel, choose any number from <b>1</b> to <b>1601</b> . For segment, the total number of points in all segments cannot exceed <b>1601</b> . A segment can have as few as 1 point.

<b>Return Type</b>	Long Integer
<b>Default</b>	201 for channel 21 for segment
<hr/>	
<b>Examples</b>	chan.NumberOfPoints = 201 'sets the number of points for all measurements in the channel. -Write numofpts = chan.NumberOfPoints 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_NumberOfPoints(long *pVal) HRESULT put_NumberOfPoints(long newVal)
<b>Interface</b>	IChannel ISegment

<b>Write/Read</b>	<b>About Power Slope</b>
<b>PowerSlope Property</b>	

<b>Description</b>	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.
<b>VB Syntax</b>	<i>app.PowerSlope = value</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>app</i>	An Application ( <b>object</b> )
<i>value</i>	<b>(double)</b> - Power Slope. Choose any number between -2 and 2. <b>No slope = 0</b>
<b>Return Type</b>	Double
<b>Default</b>	0
<hr/>	
<b>Examples</b>	app.PowerSlope = 2 'Write pwrslop = app.PowerSlope 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_PowerSlope(double *pVal) HRESULT put_PowerSlope(double newVal)
<b>Interface</b>	IChannel

<b>Write/Read</b>	<b>About Receiver Attenuation</b>
<b>ReceiverAttenuator Property</b>	

<b>Description</b>	Sets or returns the value of the specified receiver attenuator control.
<b>VB Syntax</b>	<i>chan.ReceiverAttenuator(rec) = value</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>chan</i>	A Channel ( <b>object</b> )
<i>rec</i>	<b>(long integer)</b> - Receiver with attenuator control to be changed. Choose from: <b>0</b> - Receiver A <b>1</b> - Receiver B
<i>value</i>	<b>(double)</b> - 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.
<b>Return Type</b>	Double
<b>Default</b>	0 db
<b>Examples</b>	chan.ReceiverAttenuator(1) = 5 'Write attn = chan.ReceiverAttenuator(rnum) 'Read
<b>C++ Syntax</b>	HRESULT get_ReceiverAttenuator(long lport, double *pVal) HRESULT put_ReceiverAttenuator(long lport, double newVal)
<b>Interface</b>	IChannel

**Write / Read**  
**SourcePowerCorrection Property**

**About Source Power Cal**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>chan</i> . <b>SourcePowerCorrection</b> (sourcePort) = <i>value</i>
<b>Variable</b> <i>chan</i> <i>sourcePort</i>  <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> – A Channel object <b>(long integer)</b> – Source port for which to set or return the ON or OFF state of source power correction. <b>(boolean)</b> False (0) – Turns source power correction OFF for the source port. True (1) – Turns source power correction ON for the source port.
<b>Return Type</b> <b>Default</b>	Boolean False (0) - Source power correction will turn correction ON
<b>Examples</b>	chan.SourcePowerCorrection(1) = 1 'Write calOnPort2 = chan.SourcePowerCorrection(2) 'Read
<b>C++ Syntax</b>	HRESULT put_SourcePowerCorrection(VARIANT_BOOL bState); HRESULT get_SourcePowerCorrection(VARIANT_BOOL *bState);
<b>Interface</b>	IChannel

**Write/Read**  
**StartFrequency Property**

**About Linear Frequency Sweep**

<b>Description</b>	Sets or returns the start frequency of the channel <b>or</b> Sets or returns the start frequency of the segment.
<b>VB Syntax</b>	<i>object</i> . <b>StartFrequency</b> = <i>value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel <b>(object)</b> <b>or</b>

<i>value</i>	A Segment ( <b>object</b> ) <b>(double)</b> - Start frequency in Hertz. Choose any number between the <b>minimum</b> and <b>maximum</b> frequencies of the analyzer.
<b>Return Type</b>	Double
<b>Default</b>	Channel - Minimum frequency of the analyzer Segment - 0
<hr/>	
<b>Examples</b>	chan.StartFrequency = 4.5e9 'sets the start frequency of a linear sweep for the channel object -Write startfreq = Chan.StartFrequency 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_StartFrequency(double *pVal) HRESULT put_StartFrequency(double newVal)
<b>Interface</b>	ISegment

**Write/Read**  
**StartPower Property**

**About Power Sweep**

<b>Description</b>	Sets the start power of the analyzer when sweep type is set to Power Sweep. Frequency of the measurement is set with chan.CWFrequency.
<b>VB Syntax</b>	<i>chan</i> . <b>StartPower</b> = <i>value</i>
<hr/>	
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(double)</b> - 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.
<b>Return Type</b> <b>Default</b>	Double 0
<hr/>	
<b>Examples</b>	Chan.StartPower = -10 'Write strtpwr = Chan.StartPower 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_StartPower(double *pVal) HRESULT put_StartPower(double newVal)
<b>Interface</b>	IChannel

**Write/Read**  
**StopFrequency Property**

**About Linear Frequency Sweep**

<b>Description</b>	Sets or returns the stop frequency of the channel <b>or</b> Sets or returns the stop frequency of the segment.
<b>VB Syntax</b>	<i>object</i> . <b>StopFrequency</b> = <i>value</i>
<hr/>	
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>or</b>

<i>value</i>	A Segment <b>(object)</b> <b>(double)</b> - Stop frequency in Hertz. Choose any number between the <b>minimum</b> and <b>maximum</b> frequencies of the analyzer.
<b>Return Type</b>	Double
<b>Default</b>	Channel - Maximum frequency of the analyzer Segment - 0
<b>Examples</b>	chan.StopFrequency = 4.5e9 'sets the stop frequency of a linear sweep for the channel object -Write stopfreq = Chan.StopFrequency 'Read
<b>C++ Syntax</b>	HRESULT get_StopFrequency(double *pVal) HRESULT put_StopFrequency(double newVal)
<b>Interface</b>	IChannel ISegment

### Write/Read StopPower Property

### About Power Sweep

<b>Description</b>	Sets the Stop Power of the analyzer when sweep type is set to Power Sweep. Frequency of the measurement is set with chan.CWFrequency
<b>VB Syntax</b>	<i>chan.StopPower = value</i>
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Channel <b>(object)</b> <b>(double)</b> - 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.
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	Chan.StopPower = -10 'Write stppwr = Chan.StopPower 'Read
<b>C++ Syntax</b>	HRESULT get_StopPower(double *pVal) HRESULT put_StopPower(double newVal)
<b>Interface</b>	IChannel

### Write/Read SweepGenerationMode Property

### About Stepped Sweep

<b>Description</b>	Sets the method used to generate a sweep: continuous ramp (analog) or discrete steps (stepped).
<b>VB Syntax</b>	<i>chan.SweepGenerationMode = value</i>
<b>Variable</b> <i>chan</i>	<b>(Type) - Description</b> A Channel <b>(object)</b>

*value*

**(enum NASweepGenerationModes) - Choose either:**

**0 - naSteppedSweep** - source frequency is CONSTANT during measurement of each 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  
Default**

Long Integer  
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  
SweepTime Property**

**About Sweep Time**

**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**

*chan  
value*

**(Type) - Description**

A Channel (**object**)

**(double)** - Sweep time in seconds. Choose a number between:

**0** and **100**

**Return Type  
Default**

Double  
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  
SweepType Property**

**About Sweep Types**

**Description  
VB Syntax**

Sets the type of X-axis sweep that is performed on a channel.  
*chan.SweepType = value*

**Variable**

*chan*

**(Type) - Description**

A Channel (**object**)



<i>value</i>	(enum <b>NASweepTypes</b> ) - Choose from: <b>0 - naLinearSweep</b> <b>1 - naLogSweep</b> <b>2 - naPowerSweep</b> <b>3 - naCWTimeSweep</b> <b>4 - naSegmentSweep</b> <b>Note:</b> Sweep type cannot be set to Segment sweep if there are no segments turned ON. A segment is automatically turned ON when an application is created.
<b>Return Type</b>	Long Integer
<b>Default</b>	naLinearSweep
<b>Examples</b>	chan.SweepType = naPowerSweep 'Write swptyp = chan.SweepType 'Read
<b>C++ Syntax</b>	HRESULT get_SweepType(tagNASweepTypes* pVal) HRESULT put_SweepType(tagNASweepTypes newVal)
<b>Interface</b>	IChannel

**Write/Read**  
**TestPortPower Property**

**About Power Level**

<b>Description</b>	Sets or returns the RF power level for the channel <b>or</b> Sets or returns the RF power level of the segment.
<b>VB Syntax</b>	<i>object</i> . <b>TestPortPower</b> (portNum) = value
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) - 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. <b>or</b> A Segment ( <b>object</b> )
<i>portNum</i> <i>value</i>	<b>(long integer)</b> - Port number of the source power. Choose from <b>1</b> or <b>2</b> <b>(double)</b> - RF Power in dBm. Choose any number between <b>-90 and 20</b> . Actual achievable leveled power depends on frequency.
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	chan.TestPortPower(1) = 5 'sets the port 1 RF power level for the channel object -Write powerlev = Chan.TestPortPower(1) 'Read
<b>C++ Syntax</b>	HRESULT get_TestPortPower(long port, double *pVal) HRESULT put_TestPortPower(long port, double newVal)
<b>Interface</b>	IChannel ISegment

**Write/Read**  
**TriggerMode Property**

**About Triggering**

---

<b>Description</b>	Each trigger signal will cause either: all measurements in the channel to be made <b>or</b> only a single data point in the channel at a time.
<b>VB Syntax</b>	<i>chan.TriggerMode = value</i>
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) <b>(enum NATriggerMode)</b> - Choose from: <b>0 - naTriggerModePoint</b> - 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. <b>1 - naTriggerModeMeasurement</b> - all measurements in the channel are made with each trigger signal the channel receives. <b>Note:</b> Point Mode is only available in Manual trigger and TriggerType set to naGlobalTrigger. If you change any channel to TriggerModePoint, TriggerType will be set to naChannelTrigger.
<b>Return Type</b> <b>Default</b>	Long Integer 0 - naTriggerModeMeasurement
<b>Examples</b>	<i>chan.TriggerMode = naTriggerModePoint 'Write</i> <i>trigtyp = chan.TriggerMode 'Read</i>
<b>C++ Syntax</b>	HRESULT get_TriggerMode (tagNATriggerMode *pMode) HRESULT put_TriggerMode (tagNATriggerMode newMode)
<b>Interface</b>	IChannel

**Write/Read**  
**UserRangeMax Property**

**About User Ranges**

---

<b>Description</b>	Sets the stimulus stop value for the specified User Range. This property uses different arguments for the channel and marker objects.
<b>VB Syntax</b>	<i>chan.UserRangeMax(domainType,Mnum) = value</i> <b>or</b> <i>mark.UserRangeMax(rnum) = value</i>
<b>Variable</b> <i>chan</i> <i>mark</i>  <i>domainType</i>  <i>Mnum</i>	<b>(Type) - Description</b> A Channel ( <b>object</b> ) A Marker ( <b>object</b> ) To assign a marker to a User Range, use the UserRange Property. <b>Note:</b> The Marker object does not require the "DomainType" argument. <b>(enum NADomainType)</b> - Choose from: <b>0 - naDomainFrequency</b> <b>1 - naDomainTime</b> <b>2 - naDomainPower</b> <b>(long integer)</b> - User Range number. Choose any number between 1

<i>value</i>	and 9 (0=Full Span) <b>(double)</b> - Stop value. Choose any number within the full span of the channel
<b>Return Type</b>	Double
<b>Default</b>	The current stimulus setting for the channel
<b>Examples</b>	mark.UserRangeMax(1) = 3e9 'Write chan.UserRangeMax(naDomainFrequency,1) = 3e9 'Write UseRngeMax = mark.UserRangeMax 'Read UseRngeMax = chan.UserRangeMax 'Read
<b>C++ Syntax</b>	HRESULT put_UserRangeMax(tagNADomainType domain, long rangeNumber, double maxValue) HRESULT get_UserRangeMax(tagNADomainType domain, long rangeNumber, double *maxValue)
<b>Interface</b>	IChannel

## Write/Read UserRangeMin Property

## About User Ranges

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

**Interface** rangeNumber, double \*minValue)  
IChannel

## 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

---

#### 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  
Center Property**

**About Gating**

---

<b>Description</b>	Sets or returns the Center time of either Gating or Time Domain transform windows
<b>VB Syntax</b>	<i>object.Center = value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> <b>(object)</b> As Gating <b>or</b> <b>(object)</b> As Transform <b>(double)</b> - Center time in seconds. Choose any number between: $\pm (\text{points}-1) / \text{frequency span}$
<i>value</i>	
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_Center(double *pVal) HRESULT put_Center(double newVal)
<b>Interface</b>	ITransform IGating

**Write/Read  
Shape Property**

**About Gate Filter**

---

<b>Description</b>	Specifies the shape of the gate filter.
--------------------	---

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

**Write/Read  
Span Property**

**About Time Domain**

<b>Description</b>	Sets or returns the Span time of either Gating or Time Domain transform windows
<b>VB Syntax</b>	<i>object.Span = value</i>
<b>Variable</b> <i>object</i>  <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> As Gating or <b>(object)</b> As Transform <b>(double)</b> - Span time in seconds. Choose any number between: <b>2*[(number of points-1) / frequency span]</b> and <b>0</b>
<b>Return Type</b> <b>Default</b>	Double 20ns
<b>Examples</b>	<i>Trans.Span = 4.5e-9 'sets the time span of a transform window -Write</i> <i>Gate.Span = 4.5e-9 'sets the Span time of a gating window -Write</i> <i>span = Trans.Span 'Read</i>
<b>C++ Syntax</b>	HRESULT get_Span(double *pVal) HRESULT put_Span(double newVal)
<b>Interface</b>	ITransform IGating

**Write/Read  
Start Property**

**About Time Domain**

<b>Description</b>	Sets or returns the start time of either Gating or Time Domain transform windows
--------------------	--

<b>VB Syntax</b>	<i>object.Start = value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> <b>(object)</b> As Gating <b>or</b> <b>(object)</b> As Transform
<i>value</i>	<b>(double)</b> - Start time in seconds. Choose any number between: <b>± (number of points-1) / frequency span</b>
<b>Return Type</b> <b>Default</b>	Double -10ns
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_Start(double *pVal) HRESULT put_Start(double newVal)
<b>Interface</b>	ITransform IGating

**Write/Read  
Stop Property**

**About Time Domain**

---

<b>Description</b>	Sets or returns the Stop time of either Gating or Time Domain transform windows
<b>VB Syntax</b>	<i>object.Stop = value</i>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> <b>(object)</b> As Gating <b>or</b> <b>(object)</b> As Transform
<i>value</i>	<b>(double)</b> - Start time in seconds. Choose any number between: <b>± (number of points-1) / frequency span</b>
<b>Return Type</b> <b>Default</b>	Double 10 ns
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_Stop(double *pVal) HRESULT put_Stop(double newVal)
<b>Interface</b>	ITransform IGating

**Write/Read  
Type Property**

**About Time Domain**

---

<b>Description</b>	Specifies the type of gate filter used.
<b>VB Syntax</b>	<i>gat.Type = value</i>

<b>Variable</b> <i>gat</i> <i>value</i>	<b>(Type) - Description</b> A Gating ( <b>object</b> ) <b>(enum NAGateType)</b> - Choose from: <b>0 - naGateTypeBandpass</b> - Includes (passes) the range between the start and stop times. <b>1 - naGateTypeNotch</b> - Excludes (attenuates) the range between the start and stop times.
<b>Return Type</b> <b>Default</b>	NAGateType Bandpass
<b>Examples</b>	gate.Type = naGateTypeNotch 'Write filterType = gate.Type 'Read
<b>C++ Syntax</b>	HRESULT get_Type(tagNAGateType *pVal) HRESULT put_Type(tagNAGateType newVal)
<b>Interface</b>	IGating

## HWauxIO Object

### HWAuxIO Object

#### Description

Contains the methods and properties that control the rear panel Auxiliary Input / Output connector.

See a Pinout of the Aux IO Connector

Method	Description
get_InputVoltage	Reads the ADC input voltage
get_OutputVoltage	Reads voltages on the DAC/Analog Output 1 and Output 2
get_PortCData	Reads a 4-bit value from Port C
put_OutputVoltage	Writes voltages to the DAC/Analog Output 1 and Output 2
put_PortCData	Writes a 4-bit value to Port C
Property	Description
FootSwitch	Reads the Footswitch Input
PassFailLogic	Sets and reads the logic of the PassFail line Shared with the HWMaterialHandler Object
PassFailMode	Sets and reads the mode of the PassFail line Shared with the HWMaterialHandler Object
PassFailScope	Sets and reads the scope of the PassFail line Shared with the HWMaterialHandler Object
PortCLogic	Sets and reads the logic mode of Port C
PortCMode	Sets and reads the mode of Port C
SweepEndMode	Sets and reads the event that causes the Sweep End line to go to a false state. Shared with the HWMaterialHandler Object





**Read-only  
get\_InputVoltage Method**

**About the Aux I/O Connector**

---

<b>Description</b>	Reads the ADC input voltage from Analog IN (pin 14) of the AUX IO connector
<b>VB Syntax</b>	<code>volts = AuxIO.get_InputVoltage</code>
<b>Variable</b> <i>volts</i> <i>AuxIO</i>	<b>(Type) - Description</b> <b>(double)</b> - variable to store the return value <b>(object)</b> - A Hardware Auxiliary Input / Output object
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	<pre>Dim aux as HWAuxIO Set aux = PNA.getAuxIO volts = aux.get_InputVoltage 'read voltage on Analog In (pin 14)</pre>
<b>C++ Syntax Interface</b>	HRESULT get_InputVoltage ( double* Voltage ); HWAuxIO

**Read-only  
get\_OutputVoltage Method**

**About the Aux I/O Connector**

---

<b>Description</b>	Reads voltages on the DAC/Analog Output 1 and Output 2 (pins 2 and 3 of the Aux I/O connector)
<b>VB Syntax</b>	<code>volts = AuxIO.get_OutputVoltage (output)</code>
<b>Variable</b> <i>volts</i> <i>AuxIO</i> <i>output</i>	<b>(Type) - Description</b> <b>(double)</b> - variable to store the return value <b>(object)</b> - A Hardware Auxiliary Input / Output object <b>(variant)</b> Number of the output DAC to read voltage from. Choose from:  1 - Output DAC 1 -(pin 3) 2 - Output DAC 2 -(pin 2)
<b>Return Type</b> <b>Default</b>	Double
<b>Examples</b>	<pre>Dim aux as HWAuxIO Set aux = PNA.getAuxIO volts = aux.get_OutputVoltage(1) 'read voltage from Analog Out 1 (pin3)</pre>
<b>C++ Syntax Interface</b>	HRESULT get_OutputVoltage( VARIANT Output, double* Voltage ); IHWAuxIO

**Read-only  
get\_PortCData Method**

**About the Aux I/O Connector**

<b>Description</b>	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). <b>Note:</b> These lines are connected to both the Handler IO and Aux IO in the PNA.
<b>VB Syntax</b>	<i>value</i> = <i>AuxIO.get_PortCData</i>
<b>Variable</b> <i>value</i> <i>AuxIO</i>	<b>(Type) - Description</b> <b>(variant)</b> - Variable to store the returned data <b>(object)</b> - A Hardware Auxiliary Input / Output object
<b>Return Type</b> <b>Default</b>	Integer None
<b>Examples</b>	<i>value</i> = <i>auxIo.get_PortCData</i> '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.
<b>C++ Syntax Interface</b>	HRESULT get_PortCData( VARIANT* Data ); IHWAuxIO

**Write-only  
put\_OutputVoltage Method**

**About the Aux I/O Connector**

<b>Description</b>	Writes voltages on the DAC/Analog Output 1 and Output 2 (pins 2 and 3 of the Aux I/O connector)
<b>VB Syntax</b>	<i>AuxIO.put_OutputVoltage</i> <i>output, voltage</i>
<b>Variable</b> <i>AuxIO</i> <i>output</i>	<b>(Type) - Description</b> <b>(object)</b> - A Hardware Auxiliary Input / Output object <b>(variant)</b> Number of the output DAC to write voltage to. Choose from: <b>1</b> Output DAC 1 - (pin 2) <b>2</b> Output DAC 2 - (pin 3)
<i>voltage</i>	<b>(double)</b> Voltage to write to the output DAC. Choose a voltage from -10 to 10
<b>Return Type</b> <b>Default</b>	None None
<b>Examples</b>	<i>HWAuxIO.put_OutputVoltage</i> 1,9 'set Analog Out1 to +9v
<b>C++ Syntax Interface</b>	HRESULT put_OutputVoltage (VARIANT Output, double Voltage ); IHWAuxIO

**Write-only  
put\_PortCData Method**

**About the Aux I/O Connector**

<b>Description</b>	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). <b>Note:</b> 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.
<b>VB Syntax</b>	<i>AuxIO.put_PortCData</i> <i>num</i>

<b>Variable</b> <i>AuxIO</i> <i>num</i>	<b>(Type) - Description</b> <b>(object)</b> - A Hardware Auxiliary Input / Output object <b>(variant)</b> - 4 bit binary value. Choose from 0-15
<b>Return Type</b> <b>Default</b>	None None
<b>Examples</b>	HWAuxIO.put_PortCData 15 'If Positive Logic, Port C lines C0, C1, C2, C3 go High. If Negative Logic, they go Low.
<b>C++ Syntax Interface</b>	HRESULT put_PortCData( VARIANT Data ); IHWAuxIO

### Read only FootSwitch Property

### About the Aux I/O Connector

<b>Description</b> <b>VB Syntax</b>	Reads the Footswitch Input (pin 20 of the AUX IO connector). <i>value</i> = <b>AuxIO.Footswitch</b>
<b>Variable</b> <i>value</i>	<b>(Type) - Description</b> <b>(boolean)</b> - Variable to store the returned value <b>False (0)</b> -foot switch is released <b>True (1)</b> - footswitch is depressed
<i>AuxIO</i> <b>Return Type</b> <b>Default</b>	<b>(object)</b> - A Hardware Aux I/O object Boolean True (1)
<b>Examples</b>	fs = aux.Footswitch
<b>C++ Syntax Interface</b>	HRESULT get_FootSwitch ( VARIANT_BOOL* State ); IHWAuxIO

---

### Read/Write

---

### PassFailLogic Property

<b>Description</b>	Sets and reads the logic of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12). <b>Note:</b> 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.
<b>VB Syntax</b>	<i>object</i> . <b>PassFailLogic</b> = <i>value</i>
<b>Variable</b> <i>object</i> <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> - An Aux I/O or Handler I/O object <b>(enum as NARearPanelIOLogic)</b> Choose from: <b>0 - naPositiveLogic</b> - Causes the PassFail line to have positive logic (high = pass, low = fail).

<b>Return Type</b>	<b>1 - naNegativeLogic</b> - Causes the PassFail line to have negative logic (high = fail, low = pass).
<b>Default</b>	Long Integer naPositiveLogic
<b>Examples</b>	<pre>aux.PassFailLogic = naNegativeLogic 'Write Text1.Text = aux.PassFailLogic 'Read</pre>
<b>C++ Syntax</b>	<pre>HRESULT put_PassFailLogic ( tagNARearPanellIOLogic Mode ); HRESULT get_PassFailLogic ( tagNARearPanellIOLogic* Mode );</pre>
<b>Interface</b>	IHWAuxIO IHWMaterialHandlerIO

Read/Write

### PassFailMode Property

<b>Description</b>	Sets and reads the mode of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12). <b>Note:</b> 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.
<b>VB Syntax</b>	<i>object</i> . <b>PassFailMode</b> = <i>value</i>
<b>Variable</b> <i>object</i> <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> - An Aux I/O or Handler I/O object <b>(enum as NAPassFailMode)</b> .Choose from: <b>0 - naDefaultPassNoWaitMode</b> - the line stays in PASS state. When a device fails, then the line goes to fail IMMEDIATELY. <b>1 - naDefaultPassWaitMode</b> - the line stays in PASS state. When a device fails, then the line goes to fail after the Sweep End line is asserted. <b>2 - naDefaultFailWaitMode</b> - the line stays in FAIL state. When a device passes, then the line goes to PASS state after the Sweep End line is asserted.
<b>Return Type</b> <b>Default</b>	Long Integer 0 - naDefaultPassNoWaitMode
<b>Examples</b>	<pre>HWAuxIO.PassFailMode = naDefaultPassNoWaitMode 'Write mode = HWAuxIO.PassFailMode 'Read</pre>
<b>C++ Syntax</b>	<pre>HRESULT put_PassFailMode ( tagNAPassFailMode Mode ); HRESULT get_PassFailMode ( tagNAPassFailMode* Mode );</pre>
<b>Interface</b>	IHWAuxIO IHWMaterialHandlerIO

Read/Write

### PassFailScope Property

<b>Description</b>	<p>Sets and reads the Scope of the PassFail line on the HANDLER IO connector (pin 33) and AUX IO connector (pin 12).</p> <p><b>Note:</b> 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.</p>
<b>VB Syntax</b>	<i>object.PassFailScope = value</i>
<b>Variable</b> <i>object</i> <i>value</i>	<p><b>(Type) - Description</b></p> <p><b>(object)</b> - An Aux I/O or Handler IO object</p> <p><b>(enum NAPassFailScope )</b> Choose from:</p> <p><b>0 - naChannelScope</b> - The PassFail line returns to its default state before sweeps on the next channel start. (A channel measurement may require several sweeps.)</p> <p><b>1 - naGlobalScope</b> - The PassFail line returns to its default state before the sweeps for the next <b>triggerable</b> channel start.</p> <p>The default state of the PassFail line before a measurement occurs and after a failure occurs is set by the PassFailMode property.</p>
<b>Return Type</b> <b>Default</b>	<p>enum NAPassFailScope</p> <p>1 - naGlobalScope</p>
<b>Examples</b>	<pre><b>HWAuxIO.PassFailScope = naGlobalScope 'Write</b> scope = HWAuxIO.PassFailScope 'Read</pre>
<b>C++ Syntax</b>	<pre>HRESULT put_PassFailScope ( tagNAPassFailScope Scope ); HRESULT get_PassFailScope ( tagNAPassFailScope* Scope );</pre>
<b>Interface</b>	<pre>IHWAuxIO IHWMaterialHandlerIO</pre>

## Read/Write PortCLogic Property

## About the Aux I/O Connector

<b>Description</b>	<p>Sets and reads the logic mode of Port C on the AUX IO connector and the Handler IO connector.</p> <p><b>Note:</b> 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.</p>
<b>VB Syntax</b>	<i>AuxIO.PortCLogic = value</i>
<b>Variable</b> <i>AuxIO</i> <i>value</i>	<p><b>(Type) - Description</b></p> <p><b>(object)</b> - A Hardware Aux I/O object</p> <p><b>(Enum as NaRearPanelIOLogic)</b> - Choose from:</p> <p><b>0 - naPositiveLogic</b> - The associated data line goes <b>HIGH</b> when writing a 1 to a PortC bit.</p> <p><b>1 - naNegativeLogic</b> - The associated data line goes <b>LOW</b> when writing a 1 to a PortC bit.</p> <p>When Port C is in Output/Write mode, a change in logic causes the</p>

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**  
**Default**

Enum  
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 PortCMode Property

## About the Aux I/O Connector

**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**

*AuxIO*  
*value*

**(Type) - Description**

**(object)** - A Hardware Aux I/O object

**(enum as NaPortMode)** - Choose from:

**0 - naInput** - set the port for reading

**1 - naOutput** - set the port for writing

Enum as NaPortMode

1 - naInput

**Return Type**  
**Default**

**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

## VB Syntax

will affect both of these connectors in the same way.  
*object.SweepEndMode* = *value*

## Variable

*object*  
*value*

### (Type) - Description

(object) - A HandlerIO or AuxIO object

(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 Default

Long Integer

**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

### 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  
ReadData Method**

**About the ExtTestSetIO connector**

**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 Interface** `HRESULT ReadData (VARIANT Address, VARIANT* Data);  
IHWExternaTestSetIO`

**Read-only  
ReadRaw Method**

**About the ExtTestSetIO connector**

**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)



\*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.  
*value = ExtIO.ReadRaw (address)*

**VB Syntax**

<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(variant)</b> - Variable to store the returned data
<i>ExtIO</i>	<b>(object)</b> - An External IO object
<i>address</i>	<b>(variant)</b> - Address to read data from
<b>Return Type</b>	Real
<b>Default</b>	Not Applicable

**Examples** value = ExtIO.ReadRaw (address)

**C++ Syntax Interface** HRESULT ReadRaw( VARIANT\* Input );  
 IHWExternalTestSetIO

**Write-only  
WriteData Method**

**About the ExtTestSetIO connector**

**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.  
*ExtIO.ReadData (address) = value*

**VB Syntax**

<b>Variable</b>	<b>(Type) - Description</b>
<i>ExtIO</i>	<b>(object)</b> - An External IO object
<i>address</i>	<b>(variant)</b> - address to be written to.
<i>value</i>	<b>(variant)</b> - 13-bit word to write
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

**Examples** ExtIO.WriteData (15) = 12

**C++ Syntax Interface** HRESULT WriteData(VARIANT Address, VARIANT Data);  
 IHWExternalTestSetIO

**Write-only  
WriteRaw Method**

**About the ExtTestSetIO connector**

**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** Not Applicable  
**Type**  
**Default** Not Applicable

**Examples** ExtIO.WriteRaw 12

---

**C++ Syntax Interface** HRESULT WriteRaw( VARIANT Output );  
IHWExternalTestSetIO

---

## Read-only Interrupt Property

### About the ExtTestSetIO connector

**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*

<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(boolean)</b> - Variable to store the returned data
<i>ExtIO</i>	<b>(object)</b> - An ExternalTestSetIO object
<b>Return Type</b>	Boolean
	<b>False (0)</b> - indicates the line is being held at a TTL High
	<b>True (1)</b> - indicates the line is being held at a TTL Low
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>value = ExtIO.Interrupt</code>
<b>C++ Syntax Interface</b>	<code>HRESULT get_Interrupt( VARIANT_BOOL* bValue);</code> <code>IHWExternalTestSetIO</code>

### Read-only SweepHoldOff Property

### About the ExtTestSetIO connector

---

<b>Description</b>	Returns a boolean that represents the state of SweepHoldoff line (pin2) of the External Test Set connector.
<b>VB Syntax</b>	<code>value = ExtIO.SweepHoldOff</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(boolean)</b> - Variable to store the returned data
<i>ExtIO</i>	<b>(object)</b> - An External IO object
<b>Return Type</b>	Boolean
	<b>False (0)</b> - indicates the line is being held at a TTL Low
	<b>True (1)</b> - indicates the line is being held at a TTL High
<b>Default</b>	Not Applicable
<b>Examples</b>	<code>value = ExtIO.SweepHoldOff</code>
<b>C++ Syntax Interface</b>	<code>HRESULT get_SweepHoldOff( VARIANT_BOOL* bValue);</code> <code>IHWExternalTestSetIO</code>

### HWMaterialHandlerIO Object

### HWMaterialHandlerIO Object

#### 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 get\_Input1 Method

## About the Handler IO Connector

<b>Description</b>	<p>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.</p> <p>Momentarily grounding or driving Input1 low then high causes a transition to be detected and latched.</p>
<b>VB Syntax</b>	<i>inp1</i> = handlerlo. <b>get_Input1</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>inp1</i>	<b>(variant)</b> - A variable to store the return value
<i>handlerlo</i>	<b>(object)</b> - A HandlerIO object
<b>Return Type</b>	Variant -
	<b>1</b> - a low to high transition occurred at Input1 since the last time it was queried.
	<b>0</b> - no low to high transition occurred.
<b>Default</b>	0
<b>Examples</b>	input1 = handlerlo.get_Input1 'Read
<b>C++ Syntax Interface</b>	HRESULT get_Input1 (VARIANT* Data ); IHWMaterialHandlerIO

**Read-only  
get\_Output Method**

**About the Handler IO Connector**

---

<b>Description</b>	Returns the last value written to the selected output pin. Data is written using put_Output Method
<b>VB Syntax</b>	<i>data</i> = <i>handlerIo</i> .get_Output ( <i>pin</i> )
<b>Variable</b> <i>data</i>	<b>(Type) - Description</b> <b>(variant)</b> - A variable to store the return value. The returned value will be one of the following:  <b>0</b> - TTL Low <b>1</b> - TTL High
<i>handlerIo</i> <i>pin</i>	<b>(object)</b> - A HandlerIO object <b>(enum as NAMatHandlerOutput)</b> - output pin to read value from. Choose from: <b>naOutput1 (0)</b> <b>naOutput1User (1)</b> <b>naOutput2 (2)</b> <b>naOutput2User (3)</b>
<b>Return Type</b> <b>Default</b>	Variant Not Applicable
<b>Examples</b>	<i>data</i> = <i>handlerIo</i> .get_Output( <i>naOutput1</i> )
<b>C++ Syntax</b>	HRESULT get_Output ( tagNAMatHandlerOutput Output, VARIANT* Data );
<b>Interface</b>	IHWMaterialHandlerIO

**Read-only  
get\_Port Method**

**About the Handler IO Connector**

---

<b>Description</b>	Returns the value from the specified "readable" port.				
<b>VB Syntax</b>	<i>data</i> = <i>handlerIo</i> .get_Port ( <i>port</i> )				
<b>Variable</b> <i>data</i>	<b>(Type) - Description</b> <b>(variant)</b> - A variable to store the return value. The following table shows what the returned data represents:				
	<table border="0" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;"><b>Port</b></td> <td style="text-align: center;"><b>MSB.....LSB</b></td> </tr> <tr> <td></td> <td style="text-align: center;"><b>8.....0</b></td> </tr> </table>	<b>Port</b>	<b>MSB.....LSB</b>		<b>8.....0</b>
<b>Port</b>	<b>MSB.....LSB</b>				
	<b>8.....0</b>				
	C      C3...C0				
	D      D3...D0				
	E      D3...D0 + C3...C0				
<i>handlerIo</i> <i>port</i>	<b>(object)</b> - A HandlerIO object <b>(enum as NAMatHandlerPort)</b> - port to get data from. Choose from: <b>naPortC - (2)</b> <b>naPortD - (3)</b> <b>naPortE - (4)</b>				

	<b>Note:</b> 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.
<b>Return Type</b>	Variant
<b>Default</b>	0
<hr/> <b>Examples</b>	data = handlerIo.get_Port(naPortC)
<hr/> <b>C++ Syntax Interface</b>	HRESULT get_Port ( tagNAMatHandlerPort Port, VARIANT* Data ); IHWMaterialHandlerIO

**Write-only  
put\_Output Method**

**About the Handler IO Connector**

<b>Description</b>	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).
<b>VB Syntax</b>	<i>handlerIo.put_Output (pin) = value</i>
<b>Variable</b> <i>handlerIo</i> <i>pin</i>	<b>(Type) - Description</b> <b>(object)</b> - A HandlerIO object <b>(enum as NAMatHandlerOutput)</b> - pin to write data to. Choose from: <b>naOutput1 (0)</b> - pin3 <b>naOutput1User (1)</b> - pin3 latched (applied to pin 3 on positive edge of Input1-pin2) <b>naOutput2 (2)</b> - pin4 <b>naOutput2User (3)</b> - pin4 latched (applied to pin 4 on positive edge of Input1-pin2)
<i>value</i>	<b>(Variant)</b> Value to write to the selected pin. Choose from <b>0</b> - TTL LOW <b>1</b> - TTL HIGH Not Applicable
<b>Return Type</b> <b>Default</b>	0
<hr/> <b>Examples</b>	handlerIo.put Output(naOutput1)= 1
<hr/> <b>C++ Syntax</b>	HRESULT put_Output ( tagNAMatHandlerOutput Output, VARIANT Data );
<b>Interface</b>	IHWMaterialHandlerIO

**Write-only  
put\_Port Method**

**About the Handler IO Connector**

<b>Description</b>	Writes a value to the specified port. Use the get_Port Method to read the settings from the "readable" ports (C, D, E).
<b>VB Syntax</b>	<i>handlerIo.put_Port (port) = value</i>

**Variable**  
*handlerIo*  
*port*

**(Type) - Description**  
**(object)** - A HandlerIO object  
**(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**  
**Default**

Not Applicable  
Not Applicable

**Examples**

handlerIo.put Port(naPortB)= 15

**C++ Syntax Interface**

HRESULT put\_Port ( tagNAMatHandlerPort Port, VARIANT Data );  
IHWMaterialHandlerIO

## Read/Write PortLogic Property

## About the Handler I/O Connector

**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**  
*handler*  
*value*

**(Type) - Description**  
**(object)** - A HandlerI/O object  
**(enum as NaRearPanelIOLogic)** - Choose from:

<b>Return Type</b>	Long Integer
<b>Default</b>	1 - naNegativeLogic
<b>Examples</b>	<code>handler.PortLogic = value 'Write</code> <code>value = handler.PortLogic 'Read</code>
<b>C++ Syntax</b>	HRESULT put_PortLogic( tagNARearPanelIOLogic Mode ); HRESULT get_PortLogic( tagNARearPanelIOLogic* Mode );
<b>Interface</b>	IHWMaterialHandlerIO

**Read/Write  
PortMode Property**

**About the Handler I/O Connector**

<b>Description</b>	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.
<b>VB Syntax</b>	<code>handler.PortMode (port) = value</code>
<b>Variable</b> <i>handler</i> <i>port</i>	<b>(Type) - Description</b> <b>(object)</b> - A Handler I/O object <b>(enum as NAMatHandlerPort)</b> Port to be changed. Choose from:
<i>value</i>	<b>2 -naPortC</b> <b>3- naPortD</b> <b>(enum as NaPortMode)</b> - Choose from: <b>0 - naInput</b> - set the port for reading <b>1 - naOutput</b> - set the port for writing
<b>Return Type</b> <b>Default</b>	Long Integer 1 - naInput
<b>Examples</b>	<code>handler.PortMode(naPortC) = naInput 'Write</code> <code>value = handler.PortMode(naPortD) 'Read</code>
<b>C++ Syntax</b>	HRESULT put_PortMode ( tagNAMatHandlerPort Port, tagNAPortMode Mode ); HRESULT get_PortMode ( tagNAMatHandlerPort Port, tagNAPortMode* Mode );
<b>Interface</b>	IHWMaterialHandlerIO



## IArrayTransfer Interface

### 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 <b>NACOMPLEX</b> 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 <b>NACOMPLEX</b> data into the specified buffer.
putScalar	Puts scalar data into the measurement result buffer.
Property	Description
None	



#### Read-only

#### Data Access Map

#### GetComplex Method

<b>Description</b>	Retrieves complex data from the specified location. See also getNAComplex , getData , and getPairedData Methods <i>measData.getComplex location, numPts, real(), imag()</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>measData</i> <i>location</i>	<b>(Type) - Description</b> An IArrayTransfer interface which supports the Measurement object <b>(enum NADataStore - IArrayTransfer)</b> - Where the data you want is residing. Choose from: 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor
<i>numPts</i>	See the Data Access Map <b>(long integer)</b> - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the arrays
<i>real</i>	<b>(single)</b> - Array to store the real values
<i>imag</i>	<b>(single)</b> - Array to store the imaginary values
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable

<b>Examples</b>	<pre>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)</pre>
<b>C++ Syntax</b>	IArrayTransfer - HRESULT getComplex(tagNADataStore DataStore, long* pNumValues, float* pReal, float* plmag)
<b>Interface</b>	IArrayTransfer

**Read-only Data Access Map**  
**GetNAComplex Method**

<b>Description</b>	Retrieves complex data from the specified location. See also getComplex and getData Method.
<b>VB Syntax</b>	<i>measData.getNAComplex location, numPts, data</i>
<b>Variable</b> <i>measData</i> <i>location</i>	<p><b>(Type) - Description</b>  An IArrayTransfer interface which supports the Measurement object  <b>(enum NADataStore)</b> - Where the data you want is residing. Choose from:</p> <ul style="list-style-type: none"> <li>0 - naRawData</li> <li>1 - naCorrectedData</li> <li>2 - naMeasResult</li> <li>3 - naRawMemory</li> <li>4 - naMemoryResult</li> <li>5 - naDivisor</li> </ul>
<i>numPts</i>	<p>See the Data Access Map  <b>(long integer)</b> - Number of data points requested  [out] - specifies number of data elements returned  [in] - specifies the data being requested or the capacity of the <i>dComplex</i> array</p>
<i>data</i> <b>Return Type</b> <b>Default</b>	<p><b>(NAComplex)</b> - A one-dimensional array of NaComplex to store the data.  NaComplex  Not Applicable</p>
<b>Examples</b>	<pre>Dim dComplex(201) AS NaComplex Dim measData As IArrayTransfer Dim pts as Long Set measData = app.ActiveMeasurement measData.getNAComplex naCorrectedData, pts, dComplex(0)</pre>
<b>Notes</b>	<p>The data is stored as Real and Imaginary (<b>Re</b> and <b>Im</b>) members of the NaComplex user defined type. You can access each number individually by iterating through the array.</p> <pre>For i = 0 to NumPts-1     dReal (i) = dcomplex (i).Re     dImag (i) = dcomplex (i).Im Next i</pre>
<b>C++ Syntax</b>	HRESULT getNAComplex(tagNADataStore DataStore, long* pNumValues, TsComplex* pComplex)
<b>Interface</b>	IArrayTransfer

**Read-only**  
**GetPairedData Method**

**Data Access Map**

---

<b>Description</b>	Retrieves pairs of data from the specified location. <b>Note:</b> This method exists on a non-default interface. If you cannot access this method, use the Get Data Method on IMeasurement.
<b>VB Syntax</b>	<i>measData</i> . <b>getPairedData</b> <i>location, format, numPts, d1, d2</i>
<b>Variable</b> <i>measData</i> <i>location</i>	<b>(Type) - Description</b> An IArrayTransfer interface which supports the Measurement object <b>(enum NADataStore)</b> - Where the data you want is residing. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor
<i>format</i>	See the Data Access Map <b>(enum NAPairedDataFormat)</b> - Format in which you would like the Paired data. Choose from: <ul style="list-style-type: none"><li>• <b>naLinMagPhase</b> - Linear magnitude and phase</li><li>• <b>naLogMagPhase</b> - Log magnitude and phase</li><li>• <b>naReallmaginary</b> - Real and Imaginary</li></ul> <b>Note:</b> Selecting <b>naReallmaginary</b> format is the same as using the getComplex method
<i>numPts</i>	<b>(long integer)</b> - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the <i>dPaired</i> array
<i>d1</i> <i>d2</i>	<b>(single)</b> - Array to store the magnitude / real values <b>(single)</b> - Array to store the phase / imaginary values
<b>Return Type</b> <b>Default</b>	Two Single arrays Not Applicable
<b>Examples</b>	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)
<b>C++ Syntax</b>	HRESULT getPairedData(tagNADataStore DataStore,

---

**Interface** tagNAPairedDataFormat PairFormat, long\* pNumValues, float\* pReal, float\* plmag)  
IArrayTransfer

**Read-only  
GetScalar Method**

**Data Access Map**

**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.

---

**VB Syntax**

**Note:** You can **NOT** use this command to get complex data.

*measData.getScalar location, format, numPts, data*

**Variable**

*measData*  
*location*

**(Type) - Description**

An IArrayTransfer interface which supports the Measurement object

**(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  
Default**

Single

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)
<b>C++ Syntax</b>	HRESULT getScalar(tagNADataStore DataStore, tagDataFormat DataFormat, long* pNumValues, float* pVals)
<b>Interface</b>	IArrayTransfer

**Write-only  
PutComplex Method**

**Data Access Map**

**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**

*measData*  
*location*

**(Type) - Description**

An IArrayTransfer interface which supports the Measurement object

**(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

**Return Type  
Default**

naDataFormat\_Polar

Not Applicable

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 INumValues,
float* pReal, float* plmag, tagDataFormat displayFormat)
```

**Interface**

```
IArrayTransfer
```

---

**Write-only  
PutNAComplex Method****Data Accessing Map**

---

**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**

*measData*  
*location*

**(Type) - Description**

An IArrayTransfer interface which supports the Measurement object

**(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*  
*data*

**(long integer)** - Number of data points in the channel

**(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  
Default**

Not Applicable  
Not Applicable

---

**Examples**

```
Dim measData As IArrayTransfer
Set measData = app.ActiveMeasurement
```

```
measData.putNAComplex naMemoryResult, 201, dRawComplex(0)
```

---

**C++ Syntax**

```
HRESULT putNAComplex(tagNADataStore DataStore, long INumValues,
TsComplex* pArrayOfComplex, tagDataFormat displayFormat)
```

**Interface** IArrayTransfer

**Write-only**

**Data Access Map**

**PutScalar Method**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>measData.putScalar, format, numPts, data</i>
<b>Variable</b> <i>measData</i> <i>format</i>	<b>(Type) - Description</b> An IArrayTransfer interface which supports the Measurement object. <b>(enum NADDataFormat)</b> 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 <hr/> <b>Note: Smith and Polar</b> formats are not allowed. <hr/>
<i>numPts</i>	See the Data Access Map <b>(integer)</b> - Number of values. Usually the number of points in the trace (chan.NumberOfPoints).
<i>data</i>	<b>(single)</b> - A one-dimensional array of Scalar data matching the number of points in the current measurement.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	Dim measData As IArrayTransfer Set measData = app.ActiveMeasurement  measData.putScalar naDataFormat_LogMag, 201, dScalar(0)
<b>C++ Syntax</b>	HRESULT putScalar\(\tagDataFormat eFormat, long lNumValues, float* pArrayOfScalar)
<b>Interface</b>	IArrayTransfer

**IArrayTransfer2 Interface**

**IArrayTransfer2 Interface**

---

**Description**

This interface is exactly the same as the IArrayTransfer Interface except for the following:

Wherever there is an enum "NADDataStore" argument in an IArrayTransfer method, the corresponding IArrayTransfer2 method instead uses a BSTR (string) argument. This is necessary with custom measurements which can produce buffers with names that do not have predefined

enumerations to address.

Method	Description
getComplex2	Retrieves real and imaginary data from the specified buffer.
getNAComplex2	Retrieves typed <b>NAComplex</b> data from the specified buffer.
getPairedData2	Retrieves magnitude and phase data pairs from the specified buffer.
getScalar2	Retrieves scalar data from the specified buffer.
putComplex2	Puts real and imaginary data into the specified buffer.
putNAComplex2	Puts typed <b>NAComplex</b> data into the specified buffer.
putScalar2	Puts scalar data into the measurement result buffer.
Property	Description
None	



## Read-only GetComplex2 Method

## About Custom Measurements

<b>Description</b>	Retrieves complex data from the specified location. <b>Note:</b> This method is used only for getting data from a custom measurement. To get data from a standard PNA measurement, use GetComplex Method.
<b>VB Syntax</b>	<i>measData</i> . <b>getComplex2</b> <i>location, numPts, real(), imag()</i>
<b>Variable</b> <i>measData</i>	<b>(Type) - Description</b> An IArrayTransfer2 interface which is supported by the Measurement object.
<i>location</i>	<b>(string)</b> - The name of the buffer where the data you want is residing.
<i>numPts</i>	<b>(long integer)</b> - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the arrays
<i>real</i>	<b>(single)</b> - Array to store the real values
<i>imag</i>	<b>(single)</b> - Array to store the imaginary values
<b>Return Type</b>	Single
<b>Default</b>	Not Applicable
<b>Examples</b>	Dim real(201) AS Single Dim imag(201) AS Single Dim pts as Integer Dim measData As IArrayTransfer2 Set measData = app.ActiveMeasurement measData.getComplex2 "CorrData0", pts, real(0), imag(0)0 - naRawData
<b>C++ Syntax</b>	HRESULT getComplex2(BSTR bufferName, long* pNumValues, float* pReal, float* pImag)
<b>Interface</b>	IArrayTransfer2

Read-only

About Custom Measurements



## GetNAComplex2 Method

---

<b>Description</b>	Retrieves complex data from the specified location. <b>Note:</b> This method is used only for getting data from a custom measurement. To get data from a standard PNA measurement, use GetNAComplex Method.
<b>VB Syntax</b>	<i>measData</i> . <b>getNAComplex2</b> <i>location, numPts, data</i>
<b>Variable</b> <i>measData</i>	<b>(Type) - Description</b> An IArrayTransfer2 interface which is supported by the Measurement object.
<i>location</i>	<b>(string)</b> - The name of the buffer where the data you want is residing.
<i>numPts</i>	<b>(long integer)</b> - Number of data points requested [out] - specifies number of data elements returned [in] - specifies the data being requested or the capacity of the <i>dComplex</i> array
<i>data</i>	<b>(NAComplex)</b> - A one-dimensional array of NaComplex to store the data.
<b>Return Type</b>	NAComplex
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>Dim dComplex(201) AS NaComplex Dim measData As IArrayTransfer Dim pts as Long Set measData = app.ActiveMeasurement measData.getNAComplex2 "CorrData0", pts, dComplex(0)</pre>
<b>Notes</b>	The data is stored as Real and Imaginary ( <b>Re</b> and <b>Im</b> ) 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
<b>C++ Syntax</b>	HRESULT getNAComplex2(BSTR bufferName, long* pNumValues, TsComplex* pComplex)
<b>Interface</b>	IArrayTransfer2

## Read-only GetPairedData2 Method

## About Custom Measurements

---

<b>Description</b>	Retrieves pairs of data from the specified location. <b>Note:</b> This method exists on a non-default interface. If you cannot access this method, use the Get Data Method on IMeasurement. <b>Note:</b> This method is used only for getting data from a custom measurement. To get data from a standard PNA measurement, use GetPairedData Method.
<b>VB Syntax</b>	<i>measData</i> . <b>getPairedData2</b> <i>location, format, numPts, d1, d2</i>
<b>Variable</b> <i>measData</i>	<b>(Type) - Description</b> An IArrayTransfer2 interface which is supported by the Measurement

<i>location format</i>	<p>object.</p> <p><b>(string)</b> - Name of the buffer where the data you want is residing.</p> <p><b>(enum NAPairedDataFormat)</b> - Format in which you would like the Paired data. Choose from:</p> <ul style="list-style-type: none"> <li>• <b>naLinMagPhase</b> - Linear magnitude and phase</li> <li>• <b>naLogMagPhase</b> - Log magnitude and phase</li> <li>• <b>naReallmaginary</b> - Real and Imaginary</li> </ul> <p><b>Note:</b> Selecting <b>naReallmaginary</b> format is the same as using the getComplex method</p>
<i>numPts</i>	<p><b>(long integer)</b> - Number of data points requested</p> <p>[out] - specifies number of data elements returned</p> <p>[in] - specifies the data being requested or the capacity of the <i>dPaired</i> array</p>
<i>d1</i>	<b>(single)</b> - Array to store the magnitude / real values
<i>d2</i>	<b>(single)</b> - Array to store the phase / imaginary values
<b>Return Type</b>	Two Single arrays
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	<pre>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.getPairedData2 "CorrData0", naLogMagPhase, numPoints, logm(0), phase(0)  Print values(0), values(1)</pre>
<b>C++ Syntax</b>	HRESULT getPairedData2(BSTR BufferName, tagNAPairedDataFormat PairFormat, long* pNumValues, float* pReal, float* plmag)
<b>Interface</b>	IArrayTransfer2

**Read-only  
GetScalar2 Method**

**About Custom Measurements**

<b>Description</b>	<p>Retrieves scalar data from the specified locations. You can <b>NOT</b> use this command to get complex data.</p> <hr/> <p><b>Note:</b> This method exists on a non-default interface. If you cannot access this method, use the Get Data Method on IMeasurement.</p> <hr/> <p><b>Note:</b> This method is used only for getting data from a custom measurement. To get data from a standard PNA measurement, use GetScalar Method.</p>
<b>VB Syntax</b>	<i>measData.getScalar2 location, format, numPts, data</i>
<b>Variable <i>measData</i></b>	<p><b>(Type) - Description</b></p> <p>An IArrayTransfer2 interface which supports the Measurement object.</p>

<i>location format</i>	<p><b>(string)</b> - Name of the buffer where the data you want is residing.  <b>(enum DataFormat)</b> - Scalar format in which you would like the data.  Choose from:</p> <ul style="list-style-type: none"> <li>• naDataFormat_Delay</li> <li>• naDataFormat_Imaginary</li> <li>• naDataFormat_LinMag</li> <li>• naDataFormat_LogMag</li> <li>• naDataFormat_Phase</li> <li>• naDataFormat_Real</li> </ul>
<i>numPts</i>	<p>naDataFormat_SWR  <b>(long integer)</b> - Number of data points requested  [out] - specifies number of data elements returned  [in] - specifies the data being requested or the capacity of the <i>dScalar</i> array</p>
<i>data</i> <b>Return Type</b> <b>Default</b>	<p><b>(single)</b> - Array to store the scalar data.  Single  Not Applicable</p>
<b>Examples</b>	<pre>Dim dScalar() As Single Dim measData As IArrayTransfer Set measData = app.ActiveMeasurement Dim numpts as Long numpts = app.ActiveChannel.NumberOfPoints ReDim dScalar(numPoints)  measData.getScalar2 "CorrData0", naDataFormat_LogMag, numpts, dScalar(0)  Print dScalar(0), dScalar(1)</pre>
<b>C++ Syntax</b>	<pre>HRESULT getScalar2(BSTR bufferName, tagDataFormat DataFormat, long* pNumValues, float* pVals)</pre>
<b>Interface</b>	<pre>IArrayTransfer2</pre>

**Write-only  
PutComplex2 Method**

**About Custom Measurements**

<b>Description</b>	<p>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.</p> <p><b>Note:</b> This method is used only for putting data into a custom measurement. To put data into a standard PNA measurement, use PutComplex Method</p>
<b>VB Syntax</b>	<pre><i>measData.putComplex2 location, numPts, real(), imag(), [format]</i></pre>
<b>Variable</b> <i>measData</i> <i>location</i>	<p><b>(Type) - Description</b>  An IArrayTransfer2 interface which supports the Measurement object <b>enum NADataStore</b>) Where the Data will be put. Choose from:  0 - naRawData  1 - naCorrectedData</p>

<i>numPts</i> <i>real()</i> <i>imag()</i> <i>format</i>	<p>2 - naMeasResult  3 - naRawMemory  4 - naMemoryResult  5 - naDivisor</p> <p>See the Data Access Map</p> <p><b>(long integer)</b> - Number of data points in the channel  <b>(single)</b> - Array containing real data values  <b>(single)</b> - Array containing imaginary data values  <b>(enum NADatFormat)</b> 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</p> <ul style="list-style-type: none"> <li>• naDataFormat_Delay</li> <li>• naDataFormat_Imaginary</li> <li>• naDataFormat_LinMag</li> <li>• naDataFormat_LogMag</li> <li>• naDataFormat_Phase</li> <li>• naDataFormat_Real</li> <li>• naDataFormat_SWR</li> <li>• naDataFormat_Smith</li> </ul> <p>naDataFormat_Polar  Not Applicable  Not Applicable</p>
<b>Return Type</b>	
<b>Default</b>	
<b>Examples</b>	<pre>Dim measData As IArrayTransfer Set measData = app.ActiveMeasurement  measData.putComplex2 "Memory:VectorResult0", 201, real(0),imag(0),naDataFormat_SWR</pre>
<b>C++ Syntax</b>	HRESULT putComplex2( BSTR bufferName, long INumValues, float* pReal, float* plmag, tagDataFormat displayFormat)
<b>Interface</b>	IArrayTransfer2

**Write-only  
PutNAComplex2 Method**

**About Custom Measurements**

<b>Description</b>	<p>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.</p> <p><b>Note:</b> This method is used only for putting data into a custom measurement. To put data into a standard PNA measurement, use Put NAComplex Method</p>
<b>VB Syntax</b>	<i>measData.putNAComplex2 location, numPts, data, [format]</i>
<b>Variable <i>measData</i></b>	<p><b>(Type) - Description</b>  An IArrayTransfer2 interface which supports the Measurement object.</p>

<i>location</i>	<b>(string)</b> - Name of the buffer where the data will be put.
<i>numPts</i>	<b>(long integer)</b> - Number of data points in the channel
<i>data</i>	<b>(NAComplex)</b> - A one-dimensional array of Complex data matching the number of points in the current measurement.
<i>format</i>	<b>(enum NADisplayFormat)</b> - Optional argument. Format of the data. If unspecified, naDataFormat_Polar is assumed. Only used when the destination store is naMeasResult or naMemoryResult.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>Dim measData As IArrayTransfer Set measData = app.ActiveMeasurement  measData.putNAComplex2 "Memory:VectorResult0", 201, dRawComplex(0)</pre>
<b>C++ Syntax</b>	HRESULT putNAComplex2(BSTR bufferName, long INumValues, TsComplex* pArrayOfComplex, tagDataFormat displayFormat)
<b>Interface</b>	IArrayTransfer2

## Write-only PutScalar2 Method

## About Custom Measurements

<b>Description</b>	<p>Puts Scalar data in the Measurement Result buffer. The putScalar2 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 putScalar2 data to be overwritten with the data processed from the raw data buffer.</p> <p><b>Note:</b> This method is used only for putting data into a custom measurement. To put data into a standard PNA measurement, use PutScalar Method</p>
<b>VB Syntax</b>	<i>measData.putScalar2, format, numPts, data</i>
<b>Variable</b> <i>measData</i> <i>format</i>	<p><b>(Type) - Description</b> An IArrayTransfer2 interface which supports the Measurement object.</p> <p><b>(enum NADataFormat)</b> Format of the data. Choose from:</p> <ul style="list-style-type: none"> <li>1 - naDataFormat_LinMag</li> <li>2 - naDataFormat_LogMag</li> <li>3 - naDataFormat_Phase</li> <li>6 - naDataFormat_Delay</li> <li>7 - naDataFormat_Real</li> <li>8 - naDataFormat_Imaginary</li> <li>9 - naDataFormat_SWR</li> </ul> <hr/> <p><b>Note: Smith and Polar</b> formats are not allowed.</p>
<i>numPts</i>	See the Data Access Map <b>(integer)</b> - Number of values. Usually the number of points in the trace (chan.NumberOfPoints).
<i>data</i>	<b>(single)</b> - A one-dimensional array of Scalar data matching the number of points in the current measurement.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

<b>Examples</b>	<pre>Dim measData As IArrayTransfer Set measData = app.ActiveMeasurement  measData.putScalar2 naDataFormat_LogMag, 201, dScalar(0)</pre>
<b>C++ Syntax</b>	HRESULT putScalar2(tagDataFormat eFormat, long lNumValues, float* pArrayOfScalar)
<b>Interface</b>	IArrayTransfer2

## ICalData Interface

### 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	



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

*eData*  
*term*

#### (Type) - Description

An ICalData pointer to the Calibrator object

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

- **naErrorTerm\_Directivity\_Isolation**
- **naErrorTerm\_Match**

*rcv* **naErrorTerm\_Tracking**  
**(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 Error Term	Specify these parameters:		
	<i>term</i>	<i>rcv</i>	<i>src</i>
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**  
**Default**

Single  
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\* plmag)

**Interface**

ICalData

## Write-only GetStandardComplex Method

## About Cal Sets

**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

*interface*

### (Type) - Description

An ICalData pointer to the Calibrator object **or**

An ICalData2 pointer to the Cal Set object(preferred)

*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



<i>numPts</i>	<b>(long integer)</b> - on input, max number of data points to return; on output: indicates the actual number of data points returned.
<i>real()</i>	<b>(single)</b> - array to accept the real part of the calibration data. One-dimensional for the number of data points.
<i>imag()</i>	<b>(single)</b> - array to accept the imaginary part of the calibration data. One-dimensional for the number of data points.
<b>Return Type</b>	<b>(single)</b>
<b>Default</b>	Not Applicable
<b>Examples</b>	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)
<b>C++ Syntax</b>	HRESULT getStandardComplex(tagNACalClass stdclass, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)
<b>Interface</b>	ICalData2

**Write-only** **About Accessing Data**  
**PutErrorTermComplex Method**

<b>Description</b>	Puts error term data into the error-correction data buffer. Learn more about reading and writing Cal data using COM
<b>VB Syntax</b>	<i>data.putErrorTermComplex term, rcv, src, numPts, real(), imag()</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>data</i>	An ICalData pointer to the Calibrator object
<i>term</i>	<b>(enum NAErrorTerm)</b> - The error term to be retrieved. Choose from: <ul style="list-style-type: none"> <li>• <b>naErrorTerm_Directivity_Isolation</b></li> <li>• <b>naErrorTerm_Match</b></li> </ul> <b>naErrorTerm_Tracking</b>
<i>rcv</i>	<b>(long integer)</b> - Receiver Port
<i>src</i>	<b>(long integer)</b> - Source Port
<i>numPts</i>	<b>(long integer)</b> - number of data points in the array
<i>real()</i>	<b>(single)</b> - array containing the <b>real</b> part of the calibration data. One-dimensional: the number of data points.
<i>imag()</i>	<b>(single)</b> - array containing the <b>imaginary</b> part of the calibration data. One-dimensional: the number of data points.

	To get this Error Term	Specify these parameters:		
		<i>term</i>	<i>rcv</i>	<i>src</i>
	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**  
**Default**

Not Applicable  
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* plmag)
```

**Interface**

ICalData

**Write-only**  
**PutStandardComplex Method**

**About Cal Sets**

**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**  
*interface*

**(Type) - Description**

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*

*src*

*numPts*

*real()*

*imag()*

**Return Type**

**Default**

**Examples**

**C++ Syntax**

**Interface**

**(long integer)** - Receiver Port

**(long integer)** - Source Port

**(long integer)** - number of data points in the arrays being sent.  
**(single)** - one-dimensional array containing the **real** part of the acquisition data. (0:points-1)

**(single)** - one-dimensional array containing the **imaginary** part of the acquisition data. (0:points-1)

Not Applicable

Not Applicable

Dim sdata As ICalData2

Set sdata = calmanager.CreateCal Set( 1 )

sdata.putStandardComplex naSOLT\_Open, 1, 1, numpts, rel(0), img(0)

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

ICalData

ICal Set

## ICalData2 Interface

### 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	



#### 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

*eData*  
*setID*

##### (Type) - Description

An ICalData2 pointer to the Cal Set object

**(long integer)** – specifies which error term set to read data from. (0 is the master set of etterms.)

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

*rcv* 4 - naET\_LoadMatch  
*src* 5 - naET\_Isolation  
*numPts* **(long integer)** - Receiver Port  
**(long integer)** - Source Port  
**(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 Cal Set as Cal Set
set Cal Set = pna.GetCalManager.GetCal SetByGUID( txtGUID )
Dim eData As ICalData2
Set eData = Cal Set
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\* plmag)

**Interface** ICalData2

**Write-only**  
**PutErrorTermComplex Method**

**About Cal Sets**

---

**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

<i>numPts</i>	<b>(long integer)</b> - number of data points in the array
<i>real()</i>	<b>(single)</b> - array containing the <b>real</b> part of the calibration data. One-dimensional: the number of data points.
<i>imag()</i>	<b>(single)</b> - array containing the <b>imaginary</b> part of the calibration data. One-dimensional: the number of data points.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	Dim eData As ICalData2 Set eData = app.GetCalManager.Cal Sets.Item(1) eData.putErrorTermComplex naET_LoadMatch, 1, 2, numpts, rel(0), img(0)
<b>C++ Syntax</b>	HRESULT putErrorTermComplex(tagNAErrorTerm2 ETerm, long ReceivePort, long SourcePort, long* pNumValues, float* pReal, float* pImag)
<b>Interface</b>	ICalData2

## ICalibrator2\_Interface

### 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.



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 the value of this COM property is set to

**naECALUserCharacterization1** for a particular module and that module

does not contain a user characterization, attempts to use that module will return an error until the property is set back to **naECALFactoryCharacterization**.

*interface*.ECALCharacterization(module) = value

**VB Syntax**

**Variable**

*interface*  
*module*

**(Type) - Description**

An ICalibrator2 interface (supports the Calibrator object)

**(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

enum NAECALCharacterization

naECALFactoryCharacterization

**Return Type**

**Default**

**Examples**

Dim ifcCalibrator As ICalibrator2

Dim eCharacterization As NAECALCharacterization

Set ifcCalibrator = PNAApp.ActiveChannel.Calibrator

ifcCalibrator.ECALCharacterization = naECALUserCharacterization1

'Write

eCharacterization = ifcCalibrator.ECALCharacterization 'Read

**C++ Syntax**

HRESULT put\_ECALCharacterization(tagNAECALModule  
moduleNumber, tagNAECALCharacterization characterization);

HRESULT get\_ECALCharacterization(tagNAECALModule  
moduleNumber, tagNAECALCharacterization\* characterization);

**Interface**

ICalibrator2

**INACustomMeasurement Interface**

**INACustomMeasurement Interface**

**Description**

The INACustomMeasurement interface provides the capability to manipulate the unique capabilities of a custom measurement. In addition, INACustomMeasurement interface provides access to customized data processing blocks through the GetCustomAlgorithm method.

A custom measurement is a software component that is designed to "plug-in" to the PNA software application.

See also CreateCustomMeasurement Method and Get DataByString Method.

To put and retrieve custom measurement data, use the IArrayTransfer2 Interface

Methods	Description
GetCustomAlgorithm	Retrieves a pointer to the internal custom algorithm.
GetCustomInterface	Retrieves a pointer to the internal custom measurement object
Properties	Description
None	



Read-only

About Custom Measurements

### GetCustomAlgorithm Method

<b>Description</b>	Retrieves an IUnknown interface to the specified internal custom algorithm used by this measurement object. This interface can be queried for a custom interface and subsequently used to manipulate a custom algorithm object.
<b>VB Syntax</b>	Set custom = meas. <b>GetCustomAlgorithm</b> ( <i>{guid}</i> )
<b>Variable</b> <i>custom</i> <i>meas</i> <i>{guid}</i>	<b>(Type) - Description</b> <b>(interface)</b> - IUnknown or a custom interface. <b>(object)</b> - A Measurement object (string) - GUID of your custom algorithm in GUID format
<b>Return Type</b> <b>Default</b>	IUnknown pointer Not Applicable
<b>Examples</b>	Dim custom as IMyCustomAlgorithmInterface Set custom = meas.GetCustomAlgorithm("{12345678-56D3-11D5-AD50-00108334AE98}") custom.MyCustomAlgorithmMethod 'your custom methods and properties
<b>C++ Syntax</b>	HRESULT GetCustomAlgorithm( BSTR strGUID, IUnknown** ppInterface);
<b>Interface Remarks</b>	INACustomMeasurement If the request is properly formatted and the custom algorithm requested is not found, the method returns E_NA_CUSTOM_ALGORITHM_NOT_FOUND.

Read-only

About Custom Measurements

### GetCustomInterface Method

<b>Description</b>	Retrieves an IUnknown interface to the internal custom measurement object corresponding to the measurement object on which this method is called. This interface can be queried for a custom interface and subsequently used to manipulate a custom measurement.
<b>VB Syntax</b>	Set custom = meas. <b>GetCustomInterface</b>
<b>Variable</b> <i>custom</i> <i>meas</i>	<b>(Type) - Description</b> <b>(interface)</b> - IUnknown or a custom interface. <b>(object)</b> - A Measurement object
<b>Return Type</b>	IUnknown pointer



<b>Default</b>	Not Applicable
<b>Examples</b>	Dim custom as IMyCustomMeasInterface Set custom = meas.GetCustomInterface custom.MyCustomMethod 'your custom methods and properties
<b>C++ Syntax Interface</b>	HRESULT GetCustomInterface( IUnknown** ppInterface); INACustomMeasurement
<b>Remarks</b>	If the measurement object on which the method is called is not a custom measurement (created with CreateCustomMeasurement, or corresponding front-panel operation), then the method returns E_NA_NO_CUSTOM_MEASUREMENT.

## ISourcePowerCalData Interface

### 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	



Read-only

About Source Power Cal

#### getSourcePowerCalDataScalar Method

---

<b>Description</b>	Retrieves (as scalar values) requested source power calibration data, if it exists, from this channel. <b>Note:</b> This method exists on a non-default interface. If you cannot access this method, use the getSourcePowerCalData Method on IChannel.
<b>VB Syntax</b>	<i>chandata</i> . <b>getSourcePowerCalDataScalar</b> <i>sourcePort</i> , <i>numValues</i> , <i>data</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>chandata</i>	<b>(interface)</b> – – An ISourcePowerCalData interface pointing to a Channel (object)
<i>sourcePort</i>	<b>(long integer)</b> – The source port for which calibration data is being requested.

<i>numValues</i>	<b>(long integer)</b> – 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).
<i>data</i> <b>Return Type</b> <b>Default</b>	<b>(single)</b> – Array to store the data. Single Not Applicable
<b>Examples</b>	<pre>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</pre>
<b>C++ Syntax</b>	HRESULT getSourcePowerCalDataScalar(long sourcePort, long *pNumValues, float *pVals);
<b>Interface</b>	ISourcePowerCalData

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

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

## 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 Item	Retrieves the Pass/Fail results of the Limit Test (State). 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 GetTestResult Method

### About Limit Testing

---

<b>Description</b>	Returns the result of limit line testing. There are three ways to use this command: <ul style="list-style-type: none"><li>• If neither optional parameter is specified, limit results for ALL data is returned.</li><li>• If one parameter is specified (<i>start</i>), the limit result for that data point is returned.</li></ul> If both parameters are specified, limit results are returned beginning with <i>start</i> , and ending with ( <i>start+size</i> )-1 <i>testRes = limts.GetTestResult [start,size]</i>
<b>VB Syntax</b>	

<b>Variable</b> <i>testRes</i>	<b>(Type) - Description</b> <b>(enum NALimitTestResult)</b> - A dimensioned variable to store test results
-----------------------------------	---

	0 - naLimitTestResult_None 1 - naLimitTestResult_Fail 2 - naLimitTestResult_Pass
<i>limits</i>	A LimitTest ( <b>object</b> )
<i>start</i>	( <b>long</b> ) - Optional argument. A start data point number to return limit test results.
<i>size</i>	( <b>long</b> ) - Optional argument. Number of data points from <i>start</i> to return limit test results.
<b>Return Type</b>	Long Integer
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>Dim testRes As NALimitTestResult testRes = limits.GetTestResult Select Case testRes      Case 0     Print "No Test Result"      Case 1     Print "Fails"      Case 2     Print "Pass"  End Select</pre>
<b>C++ Syntax</b>	HRESULT GetTestResult(long lStart, long lSize, tagNALimitTestResult *pVal)
<b>Interface</b>	ILimitTest

## Write/Read LineDisplay Property

## About Limits

<b>Description</b>	Turns the display of limit lines ON or OFF. To turn limit TESTING On and OFF, use State Property. <b>Note:</b> Trace data must be ON to view limit lines
<b>VB Syntax</b>	<i>limitst.LineDisplay = state</i>
<b>Variable</b> <i>limitst</i> <i>state</i>	<b>(Type) - Description</b> A LimitTest ( <b>object</b> ) <b>(boolean)</b> 0 - Turns the display of limit lines OFF 1 - Turns the display of limit lines ON
<b>Return Type</b> <b>Default</b>	Long Integer 1 - ON
<b>Examples</b>	<pre>Limtttest.LineDisplay = 1 'Write lineDsp = Limtttest.LineDisplay 'Read</pre>
<b>C++ Syntax</b>	HRESULT get_LineDisplay(VARIANT_BOOL *pVal) HRESULT put_LineDisplay(VARIANT_BOOL newVal)
<b>Interface</b>	ILimitTest

**Write/Read**  
**SoundOnFail Property**

**About Limits**

<b>Description</b>	Turns ON or OFF the audio indicator for limit failures.
<b>VB Syntax</b>	<i>limitst</i> .SoundOnFail = <i>state</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>limitst</i>	A LimitTest <b>(object)</b>
<i>state</i>	<b>(boolean)</b>
	0 - Turns the sound OFF
	1 - Turns the sound ON
<b>Return Type</b>	Long Integer
<b>Default</b>	1 - ON
<b>Examples</b>	Limttest.SoundOnFail = 1 'Write sound = Limttest.SoundOnFail 'Read
<b>C++ Syntax</b>	HRESULT get_SoundOnFail(VARIANT_BOOL *pVal) HRESULT put_SoundOnFail(VARIANT_BOOL newVal)
<b>Interface</b>	ILimitTest

**LimitSegment Object**

**LimitSegment Object**

**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**  
**BeginResponse Property**

**About Limits**

---

<b>Description</b>	When constructing a limit line, specifies the amplitude value of the start of a limit segment.
<b>VB Syntax</b>	<i>limtseg.BeginResponse = value</i>
<b>Variable</b> <i>limtseg</i> <i>value</i>	<b>(Type) - Description</b> A LimitSegment ( <b>object</b> )
<b>Return Type</b>	<b>(double)</b> - Amplitude value. No units
<b>Default</b>	Double 0
<b>Examples</b>	Set limtseg = meas.LimitTest(1) limtseg.BeginResponse = 10 'Write BegResp = limtseg.BeginResponse 'Read
<b>C++ Syntax</b>	HRESULT get_BeginResponse(double *pVal) HRESULT put_BeginResponse(double newVal)
<b>Interface</b>	ILimitSegment

**Write/Read**  
**BeginStimulus Property**

**About Limits**

---

<b>Description</b>	When constructing a limit line, specifies the beginning X-axis value.
<b>VB Syntax</b>	<i>limtseg.BeginStimulus = value</i>
<b>Variable</b> <i>limtseg</i> <i>value</i>	<b>(Type) - Description</b> A LimitSegment ( <b>object</b> )
<b>Return Type</b>	<b>(double)</b> - Stimulus value. No units
<b>Default</b>	Double 0
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_BeginStimulus(double *pVal) HRESULT put_BeginStimulus(double newVal)
<b>Interface</b>	ILimitSegment

**Write/Read**  
**EndResponse Property**

**About Limits**

---

<b>Description</b>	When constructing a limit line, specifies the amplitude value at the end of the limit segment.
<b>VB Syntax</b>	<i>limitseg.EndResponse = value</i>
<b>Variable</b> <i>limts</i> <i>value</i>	<b>(Type) - Description</b> A LimitSegment ( <b>object</b> ) <b>(double)</b> - Y-axis value of the End Response limit. No units
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	Set limitseg = meas.LimitTest(1) limitseg.EndResponse = 10 'Write EndResp = limitseg.EndResponse 'Read
<b>C++ Syntax</b>	HRESULT get_EndResponse(double *pVal) HRESULT put_EndResponse(double newVal)
<b>Interface</b>	ILimitSegment

---

<b>Write/Read</b>	<b>About Limits</b>
<b>EndStimulus Property</b>	

---

<b>Description</b>	When constructing a limit line, specifies the stimulus value for the end of the segment.
<b>VB Syntax</b>	<i>limitseg.EndStimulus = value</i>
<b>Variable</b> <i>limitseg</i> <i>value</i>	<b>(Type) - Description</b> A LimitSegment ( <b>object</b> ) <b>(double)</b> - End Stimulus X-axis value. No units
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	Set limitseg = meas.LimitTest(1) limitseg.EndStimulus = 8e9 'Write EndStim = limitseg.EndStimulus 'Read
<b>C++ Syntax</b>	HRESULT get_EndStimulus(double *pVal) HRESULT put_EndStimulus(double newVal)
<b>Interface</b>	ILimitSegment

---

<b>Write/Read</b>	<b>About Limits</b>
<b>Type (limit) Property</b>	

---

<b>Description</b>	Specifies the Limit Line type.
<b>VB Syntax</b>	<i>limt(index).Type = value</i>
<b>Variable</b> <i>limt</i> <i>index</i> <i>value</i>	<b>(Type) - Description</b> A LimitSegment ( <b>object</b> ) <b>(variant)</b> - Limit line number in the LimitTest collection <b>(enum NALimitSegmentType)</b> - Limit Line type. Choose from:

	<b>0 - naLimitSegmentType_OFF</b> - turns limit line OFF
	<b>1 - naLimitSegmentType_Maximum</b> - limit line fails with a data point ABOVE the line
	<b>2 - naLimitSegmentType_Minimum</b> - limit line fails with a data point BELOW the line
<b>Return Type</b>	Long Integer
<b>Default</b>	0 - OFF
<hr/>	
<b>Examples</b>	Set limits = meas.LimitTest limits.Type = naLimitSegmentType_Maximum 'Write limitType = limits.Type 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT put_Type(tagNALimitSegmentType *pVal) HRESULT get_Type(tagNALimitSegmentType newVal)
<b>Interface</b>	ILimitSegment

## Marker Object

### Marker Object

---

#### 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.
SearchMax	Shared with the Marker Object 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.
<b>Property</b>	<b>Description</b>
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.

<b>VB Syntax</b>	<i>object</i> . <b>Activate</b>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> Measurement ( <b>object</b> ) <b>or</b> Marker ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	meas.Activate mark.Activate
<b>C++ Syntax</b> <b>Interface</b>	HRESULT Activate() IMeasurement IMarker

**Write-only**  
**SearchMax Method**

**About Marker Search**

---

<b>Description</b> <b>VB Syntax</b>	Searches the marker domain for the maximum value. <i>mark</i> . <b>SearchMax</b>
<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	mark.SearchMax
<b>C++ Syntax</b> <b>Interface</b>	HRESULT SearchMax() IMarker

**Write-only**  
**SearchMin Method**

**About Marker Search**

---

<b>Description</b> <b>VB Syntax</b>	Searches the marker domain for the minimum value. <i>mark</i> . <b>SearchMin</b>
<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	mark.SearchMin
<b>C++ Syntax</b> <b>Interface</b>	HRESULT SearchMin() IMarker

**Write-only**  
**SearchNextPeak Method**

**About Marker Search**

<b>Description</b>	Searches the marker's domain for the next peak value.
<b>VB Syntax</b>	<i>mark</i> . <b>SearchNextPeak</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SearchNextPeak
<b>C++ Syntax</b>	HRESULT SearchNextPeak()
<b>Interface</b>	IMarker

**Write-only**  
**SearchPeakLeft Method**

**About Marker Search**

<b>Description</b>	Searches the marker's domain for the next <b>VALID</b> peak to the left of the marker.
<b>VB Syntax</b>	<i>mark</i> . <b>SearchPeakLeft</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SearchPeakLeft
<b>C++ Syntax</b>	HRESULT SearchPeakLeft()
<b>Interface</b>	IMarker

**Write-only**  
**SearchPeakRight Method**

**About Marker Search**

<b>Description</b>	Searches the marker's domain for the next <b>VALID</b> peak to the right of the marker.
<b>VB Syntax</b>	<i>mark</i> . <b>SearchPeakRight</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SearchPeakRight
<b>C++ Syntax</b>	HRESULT SearchPeakRight()
<b>Interface</b>	IMarker

**Write-only**  
**SearchTarget Method**

**About Marker Search**

<b>Description</b>	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

---

<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

---

**Examples** mark.SearchTarget

---

**C++ Syntax Interface** HRESULT SearchTarget()  
IMarker

### Write-only SearchTargetLeft Method

#### About Marker Search

---

**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

---

<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

---

**Examples** mark.SearchTargetLeft

---

**C++ Syntax Interface** HRESULT SearchTargetLeft()  
IMarker

### Write-only SearchTargetRight Method

#### About Marker Search

---

**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

---

<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable

---

**Examples** mark.SearchTargetRight

---

**C++ Syntax Interface** HRESULT SearchTargetRight()  
IMarker

### Write-only SetCenter Method

#### About Marker Functions

---

<b>Description</b>	Changes the center stimulus to the stimulus value of the marker. The start stimulus stays the same and the stop is adjusted.
<b>VB Syntax</b>	<i>mark</i> .SetCenter
<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetCenter
<b>C++ Syntax</b> <b>Interface</b>	HRESULT SetCenter() IMarker

**Write-only**  
**SetCW Method**

**About Marker Functions**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>mark</i> .SetCW
<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetCW
<b>C++ Syntax</b> <b>Interface</b>	HRESULT SetCW() IMarker

**Write-only**  
**SetElectricalDelay Method**

**About Marker Functions**

---

<b>Description</b>	Changes the measurement's electrical delay to the marker's delay value.
<b>VB Syntax</b>	<i>mark</i> .SetElectricalDelay
<b>Variable</b> <i>mark</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetElectricalDelay
<b>C++ Syntax</b> <b>Interface</b>	HRESULT SetElectricalDelay() IMarker

**Write-only**  
**SetReferenceLevel Method**

**About Marker Functions**

<b>Description</b>	Changes the measurement's reference level to the marker's Y-axis value.
<b>VB Syntax</b>	<i>mark</i> . <b>SetReferenceLevel</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetReferenceLevel
<b>C++ Syntax</b>	HRESULT SetReferenceLevel()
<b>Interface</b>	IMarker

**Write-only  
SetStart Method**

**About Marker Functions**

<b>Description</b>	Changes the start stimulus to the stimulus value of the marker. The stop stimulus stays the same and the span is adjusted.
<b>VB Syntax</b>	<i>mark</i> . <b>SetStart</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetStart
<b>C++ Syntax</b>	HRESULT SetStart()
<b>Interface</b>	IMarker

**Write-only  
SetStop Method**

**About Marker Functions**

<b>Description</b>	Changes the stop stimulus to the stimulus value of the marker. The start stimulus stays the same and the span is adjusted.
<b>VB Syntax</b>	<i>mark</i> . <b>SetStop</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>mark</i>	A Marker ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	mark.SetStop
<b>C++ Syntax</b>	HRESULT SetStop()
<b>Interface</b>	IMarker

**Write/Read  
BucketNumber Property**

**About Markers**

<b>Description</b>	Sets or returns the bucket number (data point) for the active marker.
--------------------	---

<b>VB Syntax</b>	<i>mark.BucketNumber = value</i>
<b>Variable</b> <i>mark</i> <i>value</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(long integer)</b> - 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 Long Integer
<b>Return Type</b> <b>Default</b>	The first marker is set to the middle of the span. Subsequent markers are set to the bucket number of the previously active marker.
<b>Examples</b>	<i>mark.BucketNumber = 100</i> 'moves the active marker to data point 100 - Write <i>pointNumber = mark.BucketNumber</i> 'returns the data point number the active marker is currently on. -Read
<b>C++ Syntax</b>	HRESULT get_BucketNumber(long *pVal) HRESULT put_BucketNumber(long newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**DeltaMarker Property**

**About Reference Markers**

<b>Description</b>	Sets a marker as a delta marker. The reference marker must already be turned ON. See meas.ReferenceMarkerState
<b>VB Syntax</b>	<i>mark.DeltaMarker = state</i>
<b>Variable</b> <i>app</i> <i>state</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(boolean)</b> - ON (1) marker is a delta marker OFF (0) marker is NOT a delta marker
<b>Return Type</b> <b>Default</b>	Boolean OFF (0)
<b>Examples</b>	<i>mark.DeltaMarker = True</i> 'Write <i>delta = mark.DeltaMarker</i> 'Read
<b>C++ Syntax</b>	HRESULT get_DeltaMarker(VARIANT_BOOL bState) HRESULT put_DeltaMarker(VARIANT_BOOL *bState)
<b>Interface</b>	IMarker

**Write/Read**  
**Format Property (marker)**

**About Marker Format**

<b>Description</b> <b>VB Syntax</b>	Sets (or returns) the format of the marker. <i>mark.Format = value</i>
<b>Variable</b>	<b>(Type) - Description</b>

<i>mark</i> <i>value</i>	A Marker ( <b>object</b> ) ( <b>enum NAMarkerFormat</b> ) - Choose from: <b>0</b> - naMarkerFormat_LinMag <b>1</b> - naMarkerFormat_LogMag <b>2</b> - naMarkerFormat_Phase <b>3</b> - naMarkerFormat_Delay <b>4</b> - naMarkerFormat_Real <b>5</b> - naMarkerFormat_Imaginary <b>6</b> - naMarkerFormat_SWR <b>7</b> - naMarkerFormat_LinMagPhase <b>8</b> - naMarkerFormat_LogMagPhase <b>9</b> - naMarkerFormat_Reallmaginary <b>10</b> - naMarkerFormat_ComplexImpedance <b>11</b> - naMarkerFormat_ComplexAdmittance NAMarkerFormat <b>1</b> - naMarkerFormat_LogMag
<b>Return Type Default</b>	
<b>Examples</b>	mark.Format = naMarkerFormat_SWR 'Write fmt = mark.Format 'Read
<b>C++ Syntax</b>	HRESULT get_Format(tagNAMarkerFormat *pVal) HRESULT put_Format(tagNAMarkerFormat newVal)
<b>Interface</b>	IMarker

**Write/Read Interpolated Property**

**About Markers**

<b>Description</b>	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 <b>all</b> markers in a measurement. This command will override the measurement setting. <i>mark.Interpolated = value</i>
<b>VB Syntax</b>	
<b>Variable</b> <i>mark</i> <i>value</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(boolean)</b> <b>False</b> - Turns interpolation OFF <b>True</b> - Turns interpolation ON
<b>Return Type Default</b>	Boolean True (ON)
<b>Examples</b>	mark.Interpolated = 1 'Write interpolate = mark.Interpolated 'Read
<b>C++ Syntax</b>	HRESULT get_Interpolated(VARIANT_BOOL *pVal) HRESULT put_Interpolated(VARIANT_BOOL newVal)
<b>Interface</b>	IMarker



**Read-only  
Number Property**

**About Markers**

---

**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  
PeakExcursion Property**

**About Marker Search**

---

**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  
PeakThreshold Property**

**About Marker Search**

---

**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**

<b>Return Type</b>	Single
<b>Default</b>	-100db
<hr/>	
<b>Examples</b>	mark.PeakThreshold = 1 'Write PkThresh = mark.PeakThreshold 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_PeakThreshold(float *pVal) HRESULT put_PeakThreshold(float newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**SearchFunction Property**

**About Marker Search**

<b>Description</b>	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. <i>mark.SearchFunction = value</i>
<b>VB Syntax</b>	
<hr/>	
<b>Variable</b> <i>mark</i> <i>value</i>	<b>(Type) - Description</b> A Marker <b>(object)</b> <b>(enum NAMarkerFunction)</b> - search function. Choose from: <b>0</b> - naMarkerFunction_None <b>1</b> - naMarkerFunction_Min <b>2</b> - naMarkerFunction_Max <b>3</b> - naMarkerFunction_Target <b>4</b> - naMarkerFunction_NextPeak <b>5</b> - naMarkerFunction_PeakRight <b>6</b> - naMarkerFunction_PeakLeft
<b>Return Type</b> <b>Default</b>	Long Integer <b>0</b> - naMarkerFunction_None
<hr/>	
<b>Examples</b>	mark.SearchFunction = naMarkerFunction_Target 'When this marker is set to track, it will track the Target value. searchfunction = mark.SearchFunction 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_SearchFunction(tagNAMarkerFunction *pVal) HRESULT put_SearchFunction(tagNAMarkerFunction newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**Stimulus Property**

**About Markers**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>mark.Stimulus = value</i>

<b>Variable</b> <i>mark</i> <i>value</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(double)</b> - X-Axis value. Choose any number within the full span of the channel or User Range (if set).
<b>Return Type</b> <b>Default</b>	Double 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.
<b>Examples</b>	<code>mark.Stimulus = 3e9 'Write</code> <code>XVal = mark.Stimulus 'Read</code>
<b>C++ Syntax</b>	HRESULT get_Stimulus(double *pVal) HRESULT put_Stimulus(double newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**TargetValue Property**

**About Marker Search**

---

<b>Description</b>	Sets the target value for the marker when doing Target Searches (SearchTargetLeft, SearchTarget, SearchTargetRight).
<b>VB Syntax</b>	<code>mark.TargetValue = value</code>
<b>Variable</b> <i>mark</i> <i>value</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(single)</b> - Target value. Choose any number between: <b>-500</b> and <b>500</b>
<b>Return Type</b> <b>Default</b>	Single 0
<b>Examples</b>	<code>mark.TargetValue = 10.5 'Write</code> <code>target = mark.TargetValue 'Read</code>
<b>C++ Syntax</b>	HRESULT get_TargetValue(float *pVal) HRESULT put_TargetValue(float newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**Tracking Property**

**About Marker Search**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<code>mark.Tracking = state</code>
<b>Variable</b> <i>mark</i> <i>state</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(boolean)</b> - Tracking state. Choose from:

<b>Return Type</b>	<b>ON (1)</b> <b>OFF (0)</b> Boolean 0 - Tracking OFF 1 - Tracking ON
<b>Default</b>	0 - OFF
<b>Examples</b>	mark.Tracking = 1 'Write markTracking = mark.Type 'Read
<b>C++ Syntax</b>	HRESULT put_Tracking(VARIANT_BOOL bOn) HRESULT get_Tracking(VARIANT_BOOL * pbOn)
<b>Interface</b>	IMarker

**Write/Read**  
**Type (Marker) Property**

**About Marker Types**

---

<b>Description</b>	Sets and reads the marker type.
<b>VB Syntax</b>	<i>mark.Type = value</i>
<b>Variable</b> <i>chan</i> <i>value</i>	<b>(Type) - Description</b> A Marker ( <b>object</b> ) <b>(enum NAMarkerType) - Marker Type.</b> Choose from: <b>0 - naMarkerType_Normal</b> - the X-axis value for a normal marker will always be determined by the measurement data of the marker. <b>1 - naMarkerType_Fixed</b> - retains and keeps its x-axis value at the time the marker type is set.
<b>Return Type</b>	Long Integer
<b>Default</b>	naMarkerType_Normal
<b>Examples</b>	mark.Type = naMarkerType_Normal 'Write MrkType = mark.Type 'Read
<b>C++ Syntax</b>	HRESULT get_Type(tagNAMarkerType *pVal) HRESULT put_Type(tagNAMarkerType newVal)
<b>Interface</b>	IMarker

**Write/Read**  
**UserRange Property**

**About User Ranges**

---

<b>Description</b>	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. <ul style="list-style-type: none"> <li>• Each channel has 10 user ranges.</li> <li>• Markers and trace statistics can be restricted to any user range.</li> <li>• More than one marker can occupy a user range.</li> <li>• User ranges can overlap. For example:</li> </ul>
--------------------	---

- 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.

*mark.UserRange = value*

### VB Syntax

#### Variable

*mark*

*value*

#### Return Type

**Default**

#### (Type) - Description

A Marker (**object**)

(**long integer**) - User Range. Choose any number between:

**0** and **9 (0=Full Span)**

Long Integer

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 Value Property

### About Markers

#### 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

*YValue*

*mark*

*format*

#### (Type) - Description

A variable to store the Y-axis value

A Marker (**object**)

(**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\_**Reallmaginary** (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 Interface

HRESULT get\_Value(tagNAMarkerFormat format, VARIANT \*pVal)  
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.

<i>window</i>	Only used for non-s-parameter measurements; ignored if s-parameter. <b>(long integer)</b> - Optional argument. Window number of the new measurement. Choose 1 to 4. If unspecified, the S-Parameter will be created in the Active Window.
<b>Return Type</b>	None
<b>Default</b>	None
<b>Examples</b>	meass.Add 3,"A/R1",1,1 'Adds A/R1 measurement to channel 3 in window 1
<b>C++ Syntax</b>	HRESULT Add(long ChannelNum, BSTR strParameter, long srcPort, VARIANT_BOOL bNewWindow)
<b>Interface</b>	IMeasurements

## Measurement Object

### Measurement Object

---

#### 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 its 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
```

Methods	Description
Activate	Makes an object the Active Object.
ActivateMarker	Makes a marker the Active Marker.
ChangeParameter	Changes the parameter of the measurement.
DataToDivisor	Stores data for receiver power cal of unratioded measurements
DataToMemory	Stores the active measurement into memory.
Delete	Deletes the measurement object.
DeleteAllMarkers	Deletes all of the markers from the measurement.

DeleteMarker	Deletes a marker from the active measurement.
getData	Retrieves Complex data from analyzer memory
getDataByString	Retrieves variant data from the specified location in your choice of formats.
GetFilterStatistics	Returns all four Filter Statistics
GetReferenceMarker	Returns a handle to the reference marker.
GetTraceStatistics	Returns the Trace Statistics.
InterpolateMarkers	Turns All Marker Interpolation ON and OFF for the measurement.
putDataComplex	Puts complex data into one of five data buffers.
putDataScalar	Puts formatted variant data into the measurement results buffer.
SearchFilterBandwidth	Searches the domain with the current BW target.
<b>Properties</b>	<b>Description</b>
ActiveMarker	Returns a handle to the Active Marker object.
BandwidthTarget	The insertion loss value at which the bandwidth of a filter is measured.
BandwidthTracking	Turns Bandwidth Tracking function ON and OFF.
CalibrationType	Set or get the calibration type for the measurement.
channelNumber	Returns the channel number. Shared with the Channel Object
ElectricalDelay	Sets electrical delay.
ErrorCorrection	Set or get the state of error correction for the measurement.
FilterBW	Returns the results of the SearchBandwidth method.
FilterCF	Returns the Center Frequency result of the SearchBandwidth method.
FilterLoss	Returns the Loss value of the SearchBandwidth method.
FilterQ	Returns the Q (quality factor) result of the SearchBandwidth method.
Format	Sets display format.
<b>Gating (object)</b>	
InterpolateCorrection	Turns ON and OFF the calculation of new error terms when stimulus values change.
InterpolateNormalization	Turns ON and OFF normalization interpolation when stimulus values change after receiver power cal of unratiod measurements.
<b>LimitTest (collection)</b>	
LimitTestFailed	Returns the results of limit testing
LoadPort	Returns the load port number associated with an S-parameter reflection measurement.
LogMagnitudeOffset	Sets or returns the value that normalized, unratiod, receiver power measurement data will be shifted by
<b>marker (object)</b>	
MarkerFormat	Sets or returns the format of all the markers in the measurement.
Mean	Returns the mean value of the measurement.
Name	Sets or returns the name of the measurement.
<b>NAWindow (object)</b>	
Normalization	Turns ON or OFF normalization for receiver power cal of unratiod measurements
Number	Returns the number of the measurement.
Parameter	Returns the measurement Parameter.
PeakToPeak	Returns the Peak to Peak value of the measurement.
PhaseOffset	Sets the Phase Offset for the active channel.
ReferenceMarkerState	Turns the reference marker ON or OFF
ShowStatistics	Displays and hides the measurement statistics (peak-to-peak, mean, standard deviation) on the screen.
Smoothing	Turns ON and OFF data smoothing.
SmoothingAperture	Specifies or returns the amount of smoothing as a ratio of the number of data points in the measurement trace.
StandardDeviation	Returns the standard deviation of the measurement.
StatisticsRange	Sets the User Range number for calculating measurement statistics.
<b>Trace (object)</b>	



TraceMath	Performs math operations on the measurement object and the trace stored in memory.
<b>Transform (object)</b>	
View	Sets (or returns) the type of trace displayed on the screen



**Write-only**  
**ActivateMarker Method**

**About Markers**

---

<b>Description</b>	Makes a marker the Active Marker. Use meas.ActiveMarker to read the number of the active marker.
<b>VB Syntax</b>	<i>meas.ActiveMarker(Mnum)</i>
<b>Variable</b> <i>meas</i> <i>Mnum</i>	<b>(Type) - Description</b> A Measurement <b>(object)</b> <b>(long integer)</b> - the number of the marker to make active. Choose any marker number from <b>1</b> to <b>9</b> .
<b>Return Type</b> <b>Default</b>	None Not Applicable
<b>Examples</b>	meas.ActiveMarker(1)*Write
<b>C++ Syntax</b>	HRESULT ActivateMarker(long IMarkerNumber)
<b>Interface</b>	IMeasurement
<b>Remarks</b>	Use ReferenceMarkerState to control the Reference marker.

**Write-only**  
**ChangeParameter Method**

**About Measurement Parameters**

---

<b>Description</b>	Changes the parameter of the measurement.
<b>VB Syntax</b>	<i>meas.ChangeParameter(param,IPort)</i>
<b>Variable</b> <i>meas</i> <i>param</i>	<b>(Type) - Description</b> A Measurement <b>(object)</b> <b>(string)</b> - New parameter. Choose from: <b>S11   S22   S21   S12</b> Additionally, for 3-port analyzers only: <b>S33   S13   S31   S23   S32</b>  For non-ratioed measurements: <b>A   B   R1   R2</b> <b>C</b> - 3-port analyzers only  For ratioed measurements: <b>A/B</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**

*I*Port

**(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

Not Applicable

Not Applicable

**Return Type**  
**Default**

---

**Examples**

meas.ChangeParameter "S11",1

---

**C++ Syntax**  
**Interface**

HRESULT ChangeParameter(BSTR parameter, long IPort)  
IMeasurement

**Write-only**  
**DataToDivisor Method**

**About Receiver Cal**

---

<b>Description</b>	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 unratiod 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.
<b>VB Syntax</b>	<i>meas.DataToDivisor</i>
<b>Variable</b> <i>meas</i>	<b>(Type) - Description</b> <b>(object)</b> - A Measurement object
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas.DataToDivisor</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT DataToDivisor(); IMeasurement

**Write-only**  
**DataToMemory Method**

**About Math Operations**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>meas.DataToMemory</i>
<b>Variable</b> <i>meas</i>	<b>(Type) - Description</b> A Measurement <b>(object)</b>
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas.DataToMemory</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT DataToMemory() IMeasurement

**Write-only**  
**Delete Method**

**About Measurement Parameters**

---

<b>Description</b> <b>VB Syntax</b>	Deletes the measurement. <i>meas.Delete</i>
<b>Variable</b> <i>meas</i>	<b>(Type) - Description</b> The Measurement object to delete <b>(object)</b>
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas.Delete</i>

---

<b>C++ Syntax Interface</b>	HRESULT Delete() IMeasurement
-----------------------------	----------------------------------

<b>Write-only</b>	<b>About Markers</b>
-------------------	----------------------

---

**DeleteAllMarkers Method**

---

<b>Description</b>	Deletes all of the markers from the measurement.
<b>VB Syntax</b>	<i>meas.DeleteAllMarkers</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	<b>(object)</b> - The Measurement object from which markers will be deleted.
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas.DeleteAllMarkers</i>
<b>C++ Syntax Interface</b>	HRESULT DeleteAllMarkers() IMeasurement

<b>Write-only</b>	<b>About Markers</b>
-------------------	----------------------

---

**DeleteMarker Method**

---

<b>Description</b>	Deletes a marker from the measurement.
<b>VB Syntax</b>	<i>meas.DeleteMarker(Mnum)</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	A Measurement <b>(object)</b>
<i>Mnum</i>	<b>(long)</b> - Any existing marker number in the measurement
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas.DeleteMarker(1)</i>
<b>C++ Syntax Interface</b>	HRESULT DeleteMarker(long IMarkerNumber) IMeasurement

<b>Read-only</b>	<b>About Accessing Data</b>
------------------	-----------------------------

---

**GetData Method**

---

<b>Description</b>	Retrieves variant data from the specified location in your choice of formats.
	<b>Note:</b> This method returns a variant which is less efficient than methods available on the IArrayTransfer interface.
	<b>Note:</b> If you plan to <b>Put</b> this data back into analyzer, <i>putDataComplex</i> (variant data) method requires complex, two-dimensional data. Therefore, request the data in <b>Polar</b> format.
<b>VB Syntax</b>	<i>data = meas.GetData location, format</i>

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>data</i>	Variant array to store the data.
<i>meas</i>	A Measurement ( <b>object</b> )
<i>location</i>	<b>(enum NADataStore)</b> - Where the data you want is residing. Choose from: 0 - naRawData 1 - naCorrectedData 2 - naMeasResult 3 - naRawMemory 4 - naMemoryResult 5 - naDivisor
<i>format</i>	See the Data Access Map <b>(enum NADataFormat)</b> - Format in which you would like the data. It does not have to be the displayed format. Choose from: <ul style="list-style-type: none"> <li>• naDataFormat_LinMag</li> <li>• naDataFormat_LogMag</li> <li>• naDataFormat_Phase</li> <li>• <b>naDataFormat_Polar *</b></li> <li>• <b>naDataFormat_Smith *</b></li> <li>• naDataFormat_Delay</li> <li>• naDataFormat_Real</li> <li>• naDataFormat_Imaginary</li> <li>• naDataFormat_SWR</li> </ul> <p>* <b>Specify Smith or Polar</b> formats to obtain complex data pairs, which require a two-dimensional array <b>varData (numpts, 2)</b> to accommodate both real and imaginary data.</p> <p>All scalar formats return a single dimension <b>varData(numpts)</b>.</p> <p>Variant array - automatically dimensioned to the size of the data  Not Applicable</p>
<b>Return Type</b>	
<b>Default</b>	
<b>Examples</b>	<pre>Dim varData As Variant varData = meas.GetData(naMeasResult,naDataFormat_Phase) 'Print Data For i = 0 to chan.NumberOfPoints-1   Print varData(i) Next i</pre>
<b>C++ Syntax</b>	HRESULT getData(tagNADataStore DataStore, tagDataFormat DataFormat, VARIANT *pData)
<b>Interface</b>	IMeasurement

**Read-only  
getDataByString Method**

**About Accessing Data**

<b>Description</b>	Retrieves variant data from the specified location in your choice of formats.
<b>VB Syntax</b>	<i>data = meas.getDataByString location, format</i>

<b>Variable</b> <i>data</i> <i>meas</i> <i>location</i> <i>format</i>	<p><b>(Type) - Description</b>  <b>(variant)</b> - Array to store the data.  <b>(object)</b> - A Measurement object  <b>(string)</b> – Name of the buffer to be read.  <b>(enum NADataFormat)</b> - Format in which you would like the data. It does not have to be the displayed format. Choose from:</p> <ul style="list-style-type: none"> <li>• naDataFormat_LinMag</li> <li>• naDataFormat_LogMag</li> <li>• naDataFormat_Phase</li> <li>• <b>naDataFormat_Polar</b> *</li> <li>• <b>naDataFormat_Smith</b> *</li> <li>• naDataFormat_Delay</li> <li>• naDataFormat_Real</li> <li>• naDataFormat_Imaginary</li> <li>• naDataFormat_SWR</li> </ul> <p>* <b>Specify Smith or Polar</b> formats to obtain complex data pairs, which require a two-dimensional array <b>varData (numpts, 2)</b> to accommodate both real and imaginary data.</p>
<b>Return Type</b>	All scalar formats return a single dimension <b>varData(numpts)</b> .
<b>Default</b>	Variant array Not Applicable
<b>Examples</b>	<code>meas.getDataByString "VectorResult0", naDataFormat_Phase</code>
<b>C++ Syntax</b>	<code>HRESULT getDataByString( BSTR location, tagDataFormat dataFormat, VARIANT * pData );</code>
<b>Interface</b>	IMeasurement

**Read-only**  
**GetFilterStatistics Method**

**About Marker Search**

<b>Description</b>	Returns all four Filter Statistics resulting from a SearchFilterBandwidth. To retrieve individual filter statistics, use <code>meas.FilterCF</code> , <code>meas.FilterBW</code> , <code>meas.FilterLoss</code> , <code>meas.FilterQ</code> properties.
<b>VB Syntax</b>	<code>meas.GetFilterStatistics cf,bw,loss,q</code>
<b>Variable</b> <i>meas</i> <i>cf,bw,loss,q</i>	<b>(Type) - Description</b> A Measurement <b>(object)</b> Dimensioned variables to store the returned values
<b>Return Type</b>	<b>(double)</b> <i>cf</i> <b>(single)</b> <i>bw,loss,q</i>
<b>Default</b>	Not Applicable
<b>Examples</b>	'Dimension variables Dim cf as Double Dim bw as Single Dim loss as Single Dim q as Single

meas.GetFileterStatistics cf,bw,loss,q

---

<b>C++ Syntax</b>	HRESULT GetFilterStatistics(double* centerFreq, float* bw, float* loss, float* quality)
<b>Interface</b>	IMeasurement

---

<b>Write/Read</b>	<b>About Reference Markers</b>
<b>GetReferenceMarker Method</b>	

---

<b>Description</b>	Returns a handle to the reference marker.
<b>VB Syntax</b>	<i>meas</i> .GetReferenceMarker

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	A Measurement <b>(object)</b>
<b>Return Type</b>	Object
<b>Default</b>	Not Applicable

---

<b>Examples</b>	meas.GetReferenceMarker
-----------------	-------------------------

---

<b>C++ Syntax</b>	HRESULT GetReferenceMarker(IMarker** refMarker)
<b>Interface</b>	IMeasurement

---

<b>Read-only</b>	<b>About Trace Statistics</b>
<b>GetTraceStatistics Method</b>	

---

<b>Description</b>	Returns all four Trace Statistics. To retrieve individual Trace statistics, use Mean, PeakToPeak, StandardDeviation properties. Use ShowStatistics to display the statistics of the screen.
--------------------	---

---

<b>VB Syntax</b>	<i>meas</i> .GetTraceStatistics <i>pp,mean,stdev</i>
------------------	--

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	A Measurement <b>(object)</b>
<i>pp,mean,stdev</i>	<b>(double)</b> - Dimensioned variables to store the returned values
<b>Return Type</b>	Double
<b>Default</b>	Not Applicable

---

<b>Examples</b>	'Dimension variables Dim pp As Double Dim mean As Double Dim stdv As Double meas.GetTraceStatistics pp, mean, stdv
-----------------	--

---

<b>C++ Syntax</b>	HRESULT GetTraceStatistics(double* pp, double* mean, double* stdDeviation)
-------------------	--

---

<b>Interface</b>	IMeasurement
------------------	--------------

---

<b>Write-only</b>	<b>About Markers</b>
<b>InterpolateMarkers Method</b>	

---

<b>Description</b>	Turns <b>All</b> 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.

*meas.Interpolate* = *value*

#### VB Syntax

---

#### Variable

*meas*  
*value*

#### (Type) - Description

A Measurement (**object**)  
**(boolean)**

**False** - Turns interpolation OFF for all markers in the measurement

**True** - Turns interpolation ON for all markers in the measurement

Boolean

True (ON)

#### Return Type Default

---

#### Examples

*meas.Interpolate* = 1

#### C++ Syntax Interface

HRESULT InterpolateMarkers(VARIANT\_BOOL bNewVal)  
IMeasurement

### Write-only PutDataComplex Method

### About Accessing Data

---

#### 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

*meas*  
*location*

#### (Type) - Description

A measurement (**object**)

**(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

Not Applicable

Not Applicable

#### Return Type Default

---

#### Examples

*meas.putDataComplex* *naMeasResult*, *varData*

#### C++ Syntax

HRESULT putDataComplex(tagNADataStore DataStore, VARIANT complexData)

#### Interface

IMeasurement



**Write-only**  
**PutDataScalar Method**

**About Accessing Data**

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>meas.putDataScalar format, data</i>
<b>Variable</b> <i>meas</i> <i>format</i>	<b>(Type) - Description</b> A measurement ( <b>object</b> ) <b>(enum NADDataFormat)</b> Format of the data. Choose from: 1 - naDataFormat_LinMag 2 - naDataFormat_LogMag 3 - naDataFormat_Phase <b>4 - naDataFormat_Polar *</b> <b>5 - naDataFormat_Smith *</b> 6 - naDataFormat_Delay 7 - naDataFormat_Real 8 - naDataFormat_Imaginary 9 - naDataFormat_SWR  * <b>Smith and Polar</b> formats require a two-dimensional array <b>varData (numpts, 2)</b> to accomodate both real and imaginary data. All other formats are a single dimension <b>varData(numpts)</b> . <b>(variant)</b> - A <b>two-dimensional</b> complex variant data array. <b>Note:</b> 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 <b>Polar</b> format when using the getData method.
<i>data</i>	
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	measData.putDataScalar naDataFormat_Real, varData
<b>C++ Syntax</b>	HRESULT putDataScalar(tagNADDataStore DataStore, VARIANT complexScalar)
<b>Interface</b>	IMeasurement

**Write-only**  
**SearchFilterBandwidth Method**

**About Marker Search**

---

<b>Description</b>	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.

*meas.SearchFilterBandwidth*

#### VB Syntax

---

##### Variable

*meas*

##### Return Type

Default

##### (Type) - Description

A Measurement (**object**)

Not Applicable

Not Applicable

##### Examples

---

*meas.SearchFilterBandwidth*

##### C++ Syntax

##### Interface

HRESULT SearchFilterBandwidth()

IMeasurement

#### Read-only ActiveMarker Property

#### About Markers

---

##### 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

*meas*

*<setting>*

*mark*

##### Return Type

Default

##### (Type) - Description

**(object)** - An Measurement object

A marker property (or method) and arguments

**(object)** - A marker object

marker object

None

##### Examples

---

Public mark as marker

Set mark = *meas.ActiveMarker*

##### C++ Syntax

##### Interface

HRESULT get\_ActiveMarker(IMarker\*\* marker)

IMeasurement

#### Write/Read BandwidthTarget Property

#### About Marker Search

---

##### 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**.  
*meas.BandwidthTarget = value*

**VB Syntax**

**Variable**

*meas*

*value*

**Return Type**

**Default**

**(Type) - Description**

A Measurement (**object**)

**(single)** - Target value. Choose any number between **-500** and **500**

Single

-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  
 BandwidthTracking Property**

**About Marker Search**

**Description**

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**

**Variable**

*meas*

*value*

**Return Type**

**Default**

**(Type) - Description**

A Measurement (**object**)

**(boolean)**

**1** - Turns bandwidth tracking ON

**0** - Turns bandwidth tracking OFF

Boolean

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**  
**CalibrationType Property**

**About Performing a Calibration**

---

<b>Description</b> <b>VB Syntax</b>	Specifies the type of calibration to perform or apply to the measurement. <i>meas.CalibrationType = type</i>
<b>Variable</b> <i>meas</i> <i>type</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(enum NACalType)</b> - Calibration type. Choose from: <b>0</b> - naCalType_Response_Open <b>1</b> - naCalType_Response_Short <b>2</b> - naCalType_Response_Thru <b>3</b> - naCalType_Response_Thru_And_Isol <b>4</b> - naCalType_OnePort <b>5</b> - naCalType_TwoPort_SOLT <b>6</b> - naCalType_TwoPort_TRL <b>7</b> - naCalType_None <b>8</b> - naCalType_ThreePort_SOLT
<b>Return Type</b> <b>Default</b>	<b>NACalType</b> naCalType_None
<b>Examples</b>	<i>meas.CalibrationType = naCalType_Response_Open</i> 'Write <i>meascal = meas.CalibrationType</i> 'Read
<b>C++ Syntax</b>	HRESULT put_CalibrationType (tagNACalType CalType) HRESULT get_CalibrationType (tagNACalType* pCalType)
<b>Interface</b>	IMeasurement

**Write/Read**  
**ElectricalDelay Property**

**About Electrical Delay**

---

<b>Description</b> <b>VB Syntax</b>	Sets the Electrical Delay for the active channel. <i>meas.ElectricalDelay = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(double)</b> - Electrical Delay in seconds. Choose any number between - <b>9.99</b> and <b>9.99</b>
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	<i>meas.ElectricalDelay = 1e-3</i> 'Write <i>edelay = meas.ElectricalDelay</i> 'Read
<b>C++ Syntax</b>	HRESULT get_ElectricalDelay(double *pVal) HRESULT put_ElectricalDelay(double newVal)
<b>Interface</b>	IMeasurement

**Write/Read**  
**ErrorCorrection Property**

**About Performing a Calibration**

---

<b>Description</b>	Sets (or returns) error correction ON or OFF for the measurement.
<b>VB Syntax</b>	<i>meas</i> . <b>ErrorCorrection</b> = <i>value</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	A Measurement ( <b>object</b> )
<i>value</i>	<b>(boolean)</b>
	0 - Turns error correction OFF
	1 - Turns error correction ON
<b>Return Type</b>	Boolean
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>meas</i> .ErrorCorrection = 1 'Write errcorr = <i>meas</i> .ErrorCorrection 'Read
<b>C++ Syntax</b>	HRESULT put_ErrorCorrection (VARIANT_BOOL bState) HRESULT get_ErrorCorrection (VARIANT_BOOL *bState)
<b>Interface</b>	IMeasurement

**Read-only**  
**FilterBW Property**

**About Marker Search**

---

<b>Description</b>	Returns the results of the SearchBandwidth method.
<b>VB Syntax</b>	<i>filtBW</i> = <i>meas</i> . <b>FilterBW</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>filtBW</i>	<b>(single)</b> - Variable to store bandwidth data
<i>meas</i>	A Measurement ( <b>object</b> )
<b>Return Type</b>	Single
<b>Default</b>	Not applicable
<b>Examples</b>	<i>filterBW</i> = <i>meas</i> .FilterBW 'Read
<b>C++ Syntax</b>	HRESULT get_FilterBW(float* bw)
<b>Interface</b>	IMeasurement

**Read-only**  
**FilterCF Property**

**About Marker Search**

---

<b>Description</b>	Returns the Center Frequency result of the SearchBandwidth method.
<b>VB Syntax</b>	<i>filtCF</i> = <i>meas</i> . <b>FilterCF</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>filtCF</i>	<b>(double)</b> - Variable to store bandwidth CF data
<i>meas</i>	A Measurement ( <b>object</b> )

<b>Return Type</b>	Double
<b>Default</b>	Not applicable
<b>Examples</b>	<code>filtCF = meas.FilterCF 'Read</code>
<b>C++ Syntax Interface</b>	HRESULT get_FilterCF(double* centerFrequency) IMeasurement

**Read-only  
FilterLoss Property**

**About Marker Search**

---

<b>Description</b>	Returns the Loss value of the SearchBandwidth method.
<b>VB Syntax</b>	<code>filtLoss = meas.FilterLoss</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>filtLoss</i>	<b>(single)</b> - Variable to store bandwidth Loss data
<i>meas</i>	A Measurement <b>(object)</b>
<b>Return Type</b>	Single
<b>Default</b>	Not applicable
<b>Examples</b>	<code>filterLoss = meas.FilterLoss 'Read</code>
<b>C++ Syntax Interface</b>	HRESULT get_FilterLoss(float* loss) IMeasurement

**Read-only  
FilterQ Property**

**About Marker Search**

---

<b>Description</b>	Returns the Q (quality factor) result of the SearchBandwidth method.
<b>VB Syntax</b>	<code>filtQ = meas.FilterQ</code>
<b>Variable</b>	<b>(Type) - Description</b>
<i>filtQ</i>	<b>(single)</b> - Variable to store bandwidth Q data
<i>meas</i>	A Measurement <b>(object)</b>
<b>Return Type</b>	Single
<b>Default</b>	Not applicable
<b>Examples</b>	<code>filtQ = meas.FilterQ 'Read</code>
<b>C++ Syntax Interface</b>	HRESULT get_FilterQ(float* quality) IMeasurement

**Write/Read  
Format Property**

**About Data Format**

---

<b>Description</b>	Sets or returns the display format of the measurement.
--------------------	--

<b>VB Syntax</b>	<i>meas.Format = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(enum NADataFormat)</b> - Choose from: <b>0</b> - naDataFormat_LinMag <b>1</b> - naDataFormat_LogMag <b>2</b> - naDataFormat_Phase <b>3</b> - naDataFormat_Polar <b>4</b> - naDataFormat_Smith <b>5</b> - naDataFormat_Delay <b>6</b> - naDataFormat_Double <b>7</b> - naDataFormat_Imaginary <b>8</b> - naDataFormat_SWR
<b>Return Type</b> <b>Default</b>	Long Integer 1 - naDataFormat_LogMag
<b>Examples</b>	app.TriggerMode = naTriggerModePoint 'Write fmt = meas.Format 'Read
<b>C++ Syntax</b>	HRESULT get_Format(tagDataFormat *pVal) HRESULT put_Format(tagDataFormat newVal)
<b>Interface</b>	IMeasurement

<b>Write/Read</b>	<b>About Interpolation</b>
<b>Interpolate Correction Property</b>	

<b>Description</b>	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.
<b>VB Syntax</b>	<i>meas.InterpolateCorrection = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(boolean)</b> - Choose from: <b>True</b> - Turns correction interpolation ON <b>False</b> - Turns correction interpolation OFF
<b>Return Type</b> <b>Default</b>	Boolean True
<b>Examples</b>	meas.InterpolateCorrection = False callInterpolate = InterpolateCorrection 'Read
<b>C++ Syntax</b>	HRESULT get_InterpolateCorrection(boolean *pVal) HRESULT put_InterpolateCorrection(boolean newVal)
<b>Interface</b>	IMeasurement

**Write/Read**  
**InterpolateNormalization Property**

**About Receiver Cal**

---

<b>Description</b>	<p>Turns ON and OFF normalization interpolation which calculates new divisor data when stimulus values change after normalization.</p> <p>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.</p> <p>When this property is OFF under the same circumstances, normalization is turned OFF.</p> <p>Normalization is currently supported only on measurements of unratiod power for the purpose of performing a receiver power calibration.</p> <p><i>meas.InterpolateNormalization = value</i></p>
<b>VB Syntax</b>	
<b>Variable</b> <i>meas</i> <i>value</i>	<p><b>(Type) - Description</b> <b>(object)</b> - A Measurement object <b>(boolean)</b></p> <p>0 – Turns normalization interpolation OFF 1 – Turns normalization interpolation ON</p>
<b>Return Type</b> <b>Default</b>	<p>Boolean 0 -OFF</p>
<b>Examples</b>	<p><i>meas.InterpolateNormalization = 1 'Write normalized = meas.InterpolateNormalization 'Read</i></p>
<b>C++ Syntax</b>	<p>HRESULT put_InterpolateNormalization(VARIANT_BOOL bState); HRESULT get_InterpolateNormalization(VARIANT_BOOL *bState);</p>
<b>Interface</b>	<p>IMeasurement</p>

**Read-only**  
**LimitTestFailed Property**

**About Limit Testing**

---

<b>Description</b> <b>VB Syntax</b>	<p>Returns the results of limit testing for the measurement.</p> <p><i>testFailed = meas.LimitTestFailed</i></p>
<b>Variable</b> <i>testFailed</i>	<p><b>(Type) - Description</b> <b>(boolean)</b> Variable to store the returned value</p> <p><b>False (0)</b> - Limit Test Passed <b>True (1)</b> - Limit Test Failed</p>
<i>meas</i> <b>Return Type</b> <b>Default</b>	<p>A Measurement <b>(object)</b> Boolean False returned if there is no testing in progress</p>
<b>Examples</b>	<p>Dim testRes As Boolean testRes = meas.LimitTestFailed MsgBox (testRes)</p>



---

**C++ Syntax  
Interface**

HRESULT get\_LimitTestFailed(VARIANT\_BOOL\* trueIfFailed)  
IMeasurement

**Read-only  
LoadPort Property****About Limit Testing**

---

**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**

*loadPort*

**(Type) - Description**

**(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  
Interface**

HRESULT get\_LoadPort(long \*pPortNumber);  
IMeasurement

**Write/Read  
LogMagnitudeOffset Property****About Receiver Cal**

---

**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**

*meas*

*value*

**(Type) - Description**

**(object)** - A Measurement object

**(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**  
**MarkerFormat Property**

**About Marker Format**

---

**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\_Reallmaginary  
**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**  
**Mean Property**

**About Trace Statistics**

---

**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**

<i>average</i>	<b>(single)</b> - Variable to store mean value
<i>meas</i>	A Measurement <b>(object)</b>
<b>Return Type</b>	Single
<b>Default</b>	Not applicable
<hr/>	
<b>Examples</b>	Dim average as Single average = meas.Mean 'Read
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT get_Mean(float* mean) IMeasurement

<b>Write/Read</b>	<b>About Receiver Cal</b>
<b>Normalization Property</b>	

<b>Description</b>	Sets or returns normalization ON or OFF for the measurement. Normalization is currently supported only on measurements of unratiod 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.
<b>VB Syntax</b>	<i>meas.Normalization = value</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	<b>(object)</b> - A Measurement object
<i>value</i>	<b>(boolean)</b>
	0 – Turns normalization OFF
	1 – Turns normalization ON
<b>Return Type</b>	Boolean
<b>Default</b>	0 -OFF
<hr/>	
<b>Examples</b>	meas.Normalization = 1 'Write normalized = meas.Normalization 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT put_Normalization(VARIANT_BOOL bState); HRESULT get_Normalization(VARIANT_BOOL *bState);
<b>Interface</b>	IMeasurement

<b>Write/Read</b>	<b>About Traces</b>
<b>Name (Measurement) Property</b>	

<b>Description</b>	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. <b>Note:</b> This is the same name as trace.Name; when one changes, the other changes.
<b>VB Syntax</b>	<i>meas.Name = value</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>

<i>meas</i>	A Measurement ( <b>object</b> )
<i>value</i>	<b>(string)</b> - A user defined name of the measurement
<b>Return Type</b>	String
<b>Default</b>	"CH1_S11_1" - name of the default measurement
<hr/>	
<b>Examples</b>	meas.Name = "Filter BPass" 'Write MName = meas.Name 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_Name(BSTR *pVal) HRESULT put_Name(BSTR newVal)
<b>Interface</b>	IMeasurement

## Read-only Number (Measurement) Property About Measurements

<b>Description</b>	Returns the Number of the measurement. Measurement numbers are assigned internally.
<hr/>	
<b>Note:</b> 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.	
<hr/>	
This property is used to identify measurements when events occur through the OnMeasurementEvent callback. For example: OnMeasurementEvent (naEventId_MSG_LIMIT_FAILED, 3) <i>measNum</i> = <i>meas</i> . <b>Number</b>	
<b>VB Syntax</b>	
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>measNum</i>	<b>(long)</b> - variable to store the measurement number
<i>meas</i>	A Measurement ( <b>object</b> )
<b>Return Type</b>	Long Integer
<b>Default</b>	"1" - number of the default measurement
<hr/>	
<b>Examples</b>	measNum = meas.Number
<hr/>	
<b>C++ Syntax</b>	HRESULT get_Number(long *MeasurementNumber)
<b>Interface</b>	IMeasurement

## Read-only Parameter Property

<b>Description</b>	Returns the measurement Parameter. To change the parameter, use <i>meas</i> .ChangeParameter
<b>VB Syntax</b>	<i>measPar</i> = <i>meas</i> . <b>Parameter</b>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>measPar</i>	<b>(string)</b> - Variable to store Parameter string
<i>meas</i>	A Measurement ( <b>object</b> )
<b>Return Type</b>	String
<b>Default</b>	Not applicable

---

<b>Examples</b>	measPar = meas.Parameter 'Read
-----------------	--------------------------------

---

<b>C++ Syntax Interface</b>	HRESULT get_Parameter(BSTR *pVal) IMeasurement
-----------------------------	---

---

**Read-only**  
**PeakToPeak Property**

**About Trace Statistics**

---

<b>Description</b>	Returns the Peak to Peak value of the measurement. To retrieve all 3 statistics value at the same time, use meas.GetTraceStatistics
--------------------	---

---

<b>VB Syntax</b>	<i>pp</i> = meas. <b>PeakToPeak</b>
------------------	-------------------------------------

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>pp</i>	<b>(single)</b> - Variable to store peak-to-peak value
<i>meas</i>	A Measurement <b>(object)</b>
<b>Return Type</b>	Single
<b>Default</b>	Not applicable

---

<b>Examples</b>	pp = meas.PeakToPeak 'Read
-----------------	----------------------------

---

<b>C++ Syntax Interface</b>	HRESULT get_PeakToPeak(float* pp) IMeasurement
-----------------------------	---

---

**Write/Read**  
**PhaseOffset Property**

**About Phase Offset**

---

<b>Description</b>	Sets the Phase Offset for the active channel.
<b>VB Syntax</b>	<i>meas.PhaseOffset</i> = <i>value</i>

---

<b>Variable</b>	<b>(Type) - Description</b>
<i>meas</i>	A Measurement <b>(object)</b>
<i>value</i>	<b>(double)</b> - PhaseOffset in degrees. Choose any number between: <b>-360</b> and <b>+360</b>

---

<b>Return Type</b>	Double
<b>Default</b>	0

---

<b>Examples</b>	meas.PhaseOffset = 25 'Write poffset = meas.PhaseOffset 'Read
-----------------	--

---

<b>C++ Syntax Interface</b>	HRESULT get_PhaseOffset(double *pVal) HRESULT put_PhaseOffset(double newVal) IMeasurement
-----------------------------	---

---

**Write/Read**  
**ReferenceMarkerState Property**

**About Reference Markers**

---

<b>Description</b> <b>VB Syntax</b>	Turn ON or OFF the reference marker. (can you access marker10?) <i>meas.ReferenceMarkerState = state</i>
<b>Variable</b> <i>app</i> <i>state</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) (boolean) - ON (1) turns the reference marker ON OFF (0) turns the reference marker OFF
<b>Return Type</b> <b>Default</b>	Boolean 0 - OFF
<b>Examples</b>	<i>meas.ReferenceMarkerState = True</i> <i>reference = meas.ReferenceMarkerState</i>
<b>C++ Syntax</b>	HRESULT get_ReferenceMarkerState(VARIANT_BOOL bState) HRESULT put_ReferenceMarkerState(VARIANT_BOOL* bState)
<b>Interface</b>	IMeasurement

---

<b>Write/Read</b>	<b>About Trace Statistics</b>
<b>ShowStatistics Property</b>	

---

<b>Description</b>	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.
<b>VB Syntax</b>	<i>meas.ShowStatistics = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(boolean)</b> - Boolean value: 1 - Show statistics 0 - Hide statistics
<b>Return Type</b> <b>Default</b>	Boolean 0 - Hide
<b>Examples</b>	<i>meas.ShowStatistics = True 'Write</i> <i>showstats = meas.ShowStatistics 'Read</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT put_ShowStatistics(VARIANT_BOOL bState) IMeasurement

<b>Write/Read</b>	<b>About Smoothing</b>
<b>SmoothingAperture Property</b>	

---

<b>Description</b>	Specifies or returns the amount of smoothing as a ratio of the number of data points in the measurement trace.
--------------------	--

<b>VB Syntax</b>	<i>meas.SmoothingAperture = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(double)</b> - Smoothing Aperture. A ratio of (aperture points / trace points)/100 Choose any number between <b>.01</b> and <b>.25</b> .
<b>Return Type</b> <b>Default</b>	Double .25
<b>Examples</b>	<i>meas.SmoothingAperture = .10</i> 'Write <i>aperture = meas.SmoothingAperture</i> 'Read
<b>C++ Syntax</b>	HRESULT get_SmoothingAperture(double *pVal) HRESULT put_SmoothingAperture(double newVal)
<b>Interface</b>	IMeasurement

### Write/Read Smoothing Property

### About Smoothing

---

<b>Description</b> <b>VB Syntax</b>	Turns ON and OFF data smoothing. <i>meas.Smoothing = state</i>
<b>Variable</b> <i>meas</i> <i>state</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(boolean)</b> <b>1</b> - Turns smoothing ON <b>0</b> - Turns smoothing OFF
<b>Return Type</b> <b>Default</b>	Boolean 0
<b>Examples</b>	<i>meas.Smoothing = 1</i> 'Write <i>smooth = meas.Smoothing</i> 'Read
<b>C++ Syntax</b>	HRESULT get_Smoothing(VARIANT_BOOL *pVal) HRESULT put_Smoothing(VARIANT_BOOL newVal)
<b>Interface</b>	IMeasurement

### Read-only StandardDeviation Property

### About Trace Statistics

---

<b>Description</b>	Returns the standard deviation of the measurement. To retrieve all 3 statistics value at the same time, use <i>meas.GetTraceStatistics</i>
<b>VB Syntax</b>	<i>stdev = meas.StandardDeviation</i>
<b>Variable</b> <i>stdev</i> <i>meas</i>	<b>(Type) - Description</b> <b>(single)</b> - Variable to store standard deviation value A Measurement ( <b>object</b> )
<b>Return Type</b>	Single

<b>Default</b>	Not applicable
<b>Examples</b>	stdev = meas.StandardDeviation 'Read
<b>C++ Syntax Interface</b>	HRESULT get_StandardDeviation(float* stdDeviation) IMeasurement

**Write/Read**  
**Statistics Range Property**

**About User Ranges**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>meas.StatisticsRange = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A Measurement ( <b>object</b> ) <b>(long integer)</b> - Range Number. Choose any number between 0 and 9. <b>1 - 9</b> are user-defined ranges <b>0</b> is Full Span
<b>Return Type</b> <b>Default</b>	Long Integer 0
<b>Examples</b>	meas.StatisticsRange = 2 'Write statrang = meas.StatisticsRange 'Read
<b>C++ Syntax</b>	HRESULT get_StatisticsRange(long* rangeNumber) HRESULT put_StatisticsRange(long rangeNumber)
<b>Interface</b>	IMeasurement

**Write/Read**  
**TraceMath Property**

**About Math Operations**

<b>Description</b>	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.)
<b>VB Syntax</b>	<i>meas.TraceMath = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A measurement ( <b>object</b> ) <b>(enum NAMathOperation)</b> - Choose from: <b>0</b> - naDataNormal <b>1</b> - naDataMinusMemory <b>2</b> - naDataPlusMemory <b>3</b> - naDataDivMemory <b>4</b> - naDataTimesMemory
<b>Return Type</b>	NAMathOperation



<b>Default</b>	Normal (0)
<b>Examples</b>	meas.TraceMath = naDataMinusMemory 'Write mathOperation = meas.TraceMath 'Read
<b>C++ Syntax</b>	HRESULT get_TraceMath(tagNAMathOperation* pMathOp) HRESULT put_TraceMath(tagNAMathOperation mathOp)
<b>Interface</b>	IMeasurement

## Write/Read View Property

## About Math Operations

<b>Description</b> <b>VB Syntax</b>	Sets (or returns) the type of trace displayed on the screen. <i>meas.View = value</i>
<b>Variable</b> <i>meas</i> <i>value</i>	<b>(Type) - Description</b> A measurement <b>(object)</b> <b>(enum NAView) - Type of trace. Choose from:</b> <b>0</b> - naData <b>1</b> - naDataAndMemory <b>2</b> - naMemory <b>3</b> - naNoTrace <b>Note:</b> The <b>naData</b> trace may reflect the result of a TraceMath operation.
<b>Return Type</b> <b>Default</b>	NAView naData
<b>Examples</b>	meas.View = naData 'Write trceview = meas.View 'Read
<b>C++ Syntax</b>	HRESULT get_View(tagNAView* pView) HRESULT put_View(tagNAView newView)
<b>Interface</b>	IMeasurement

## 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**  
**Add (NAWindows) Method**

**About Windows**

<b>Description</b>	Add a window to the display. Does not add a measurement. The window number must not already exist.
<b>VB Syntax</b>	<i>wins.Add [item]</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>wins</i>	A NAWindow collection <b>(object)</b>
<i>item</i>	<b>(variant)</b> - optional argument; Window number. Range between 1 - 4
<b>Return Type</b>	Object
<b>Default</b>	Not Applicable
<b>Examples</b>	wins.Add 3 'Creates a window number 3
<b>C++ Syntax</b>	HRESULT Add(long windowNumber )
<b>Interface</b>	INAWindows

**NAWindow Object**  
**NAWindow Object**

**Description**

The NAWindow object controls the part of the display that contains the graticule, or what is written on the display.

<b>Methods</b>	<b>Description</b>
Autoscale	Autoscales all measurements in the window.
ShowMarkerReadout	Shared with the Trace Object 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.
<b>Property</b>	<b>Description</b>
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.
OneMarkerReadoutPerTrace	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.
<b>Traces (collection)</b>	

WindowNumber	Reads the number of the active window.
WindowState	Maximizes or minimizes a window. Shared with the Application Object



**Write-only  
Autoscale Method**

**About Display Formatting**

---

<b>Description</b>	Autoscales the trace (Trace object) or all of the traces (NAWindow object).
<b>VB Syntax</b>	<i>object</i> . <b>Autoscale</b>
<b>Variable</b> <i>object</i>	<b>(Type) - Description</b> Trace ( <b>object</b> ) <b>or</b> NAWindow ( <b>object</b> )
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	Trac.Autoscale 'Autoscales the trace Win.Autoscale 'Autoscales all the traces in the window -Write
<b>C++ Syntax</b> <b>Interface</b>	HRESULT AutoScale() INAWindow ITrace

**Write-only  
ShowMarkerReadout Method**

**About Display Formatting**

---

<b>Description</b>	Shows and Hides the Marker readout for the active marker in the upper-right corner of the window.
<b>VB Syntax</b>	<i>win</i> . <b>ShowMarkerReadout</b> <i>state</i>
<b>Variable</b> <i>win</i> <i>state</i>	<b>(Type) - Description</b> A NAWindow ( <b>object</b> ) <b>(boolean) -</b> <b>True (1)</b> - Show the Marker readout <b>False (0)</b> - Hide the Marker readout
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	win.ShowMarkerReadout True
<b>C++ Syntax</b> <b>Interface</b>	HRESULT ShowMarkerReadout(VARIANT_BOOL bState) INAWindow

**Write-only**

**About Display Formatting**

## ShowTable Method

---

<b>Description</b>	Shows or Hides the specified table for the window's active measurement in the lower part of the window.
<b>VB Syntax</b>	<i>win.ShowTable value</i>
<b>Variable</b> <i>win</i> <i>value</i>	<b>(Type) - Description</b> A NAWindow ( <b>object</b> ) <b>(enum naTable)</b> - The table to show or hide. Choose from: 0 - naTable_None 1 - naTable_Marker 2 - naTable_Segment 3 - naTable_Limit
<b>Return Type</b> <b>Default</b>	Not Applicable Not Applicable
<b>Examples</b>	<i>win.ShowTable naTable_limit</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT ShowTable (tagNATableType table) INAWindow

## Read-only ActiveTrace Property

## About Traces

---

<b>Description</b>	Returns a handle to the Active Trace object. You can either <b>(1)</b> use the handle directly to access trace properties and methods, or <b>(2)</b> set a variable to the trace object. The variable retains a handle to the original trace if another trace becomes active.
<b>VB Syntax</b>	1) <i>win.ActiveTrace.&lt;setting&gt;</i> <b>or</b> 2) Set <i>trce = win.ActiveTrace</i>
<b>Variable</b> <i>trce</i> <i>win</i> <i>&lt;setting&gt;</i>	<b>(Type) - Description</b> A Trace ( <b>object</b> ) An NAWindow ( <b>object</b> ) A trace property (or method) and arguments
<b>Return Type</b> <b>Default</b>	An NAWindow object None
<b>Examples</b>	1) <i>win.ActiveTrace.Autoscale</i> 2) Public <i>trce</i> as Trace Set <i>trce = Application.ActiveNAWindow.ActiveTrace</i>
<b>C++ Syntax</b> <b>Interface</b>	HRESULT get_ActiveTrace(ITrace* *pVal) INAWindow

## Write/Read MarkerReadout Property

## About Marker Readout

---

<b>Description</b>	Enables or disables the readout of markers in the window. To show the
--------------------	---

<b>VB Syntax</b>	marker on the screen use ShowMarkerReadout Method. <i>win.MarkerReadout = state</i>
<b>Variable</b> <i>win</i> <i>state</i>	<b>(Type) - Description</b> A NAWindow (object) <b>(boolean)</b> <b>True</b> (1) - enables marker readout <b>False</b> (0) - disables marker readout
<b>Return Type</b> <b>Default</b>	Boolean True
<b>Examples</b>	<pre>win.MarkerReadout = True 'Write State = app.ActiveNAWindow.MarkerReadout 'Read</pre>
<b>C++ Syntax</b>	HRESULT get_MarkerReadout(VARIANT_BOOL *pVal) HRESULT put_MarkerReadout(VARIANT_BOOL newVal)
<b>Interface</b>	INAWindow

<b>Write/Read</b>	<b>About Marker Readout</b>
<b>MarkerReadoutSize Property</b>	

<b>Description</b>	Specifies the size of font used when displaying Marker Readout in the selected window.
<b>VB Syntax</b>	<i>win.MarkerReadoutSize = value</i>
<b>Variable</b> <i>win</i> <i>value</i>	<b>(Type) - Description</b> A NAWindow (object) <b>(enum NAFontSize)</b> <b>0 - naDefault</b> - marker readout appears in default font size <b>1 - naLarge</b> - marker readout appears in large font size
<b>Return Type</b> <b>Default</b>	Long Integer naDefault
<b>Examples</b>	<pre>win.MarkerReadoutSize = naDefault 'write default size for marker readout Dim Size As NAFontSize Size = app.ActiveNAWindow.MarkerReadoutSize 'Read</pre>
<b>C++ Syntax</b>	HRESULT get_MarkerReadoutSize(tagNAFontSize *pVal) HRESULT put_MarkerReadoutSize(tagNAFontSize newVal)
<b>Interface</b>	INAWindow

<b>Write/Read</b>	<b>About Marker Readout</b>
<b>OneReadoutPerTrace Property</b>	

<b>Description</b>	Either show marker readout of only the active trace or all of the traces
--------------------	--

<b>VB Syntax</b>	simultaneously. <i>win.OneReadoutPerTrace = state</i>
<b>Variable</b> <i>win</i> <i>value</i>	<b>(Type) - Description</b> A NAWindow (object) <b>(boolean)</b> <b>True</b> (1) - show a single marker per trace <b>False</b> (0) - show up to 4 markers per active trace
<b>Return Type</b> <b>Default</b>	Boolean False (0)
<b>Examples</b>	<pre>win.OneReadoutPerTrace = True 'Write State = app.ActiveNAWindow.OneReadoutPerTraceBegResp 'Read</pre>
<b>C++ Syntax</b>	HRESULT get_OneReadoutPerTrace(VARIANT_BOOL *pVal) HRESULT put_OneReadoutPerTrace(VARIANT_BOOL newVal)
<b>Interface</b>	INAWindow

**Write/Read  
Title Property**

**About Title**

<b>Description</b>	Writes or reads a custom title for the window. Newer entries replace (not append) older entries. Turn the title ON and OFF with TitleState
<b>VB Syntax</b>	<i>win.Title = string</i>
<b>Variable</b> <i>win</i> <i>string</i>	<b>(Type) - Description</b> A NaWindow ( <b>object</b> ) <b>(long)</b> - Title limited to 50 characters.
<b>Return Type</b> <b>Default</b>	String Null
<b>Examples</b>	<pre>win.Title = "Hello World" 'Write titl = win.Title 'Read</pre>
<b>C++ Syntax</b>	HRESULT get_Title(BSTR *title) HRESULT put_Title(BSTR title)
<b>Interface</b>	INAWindow

**Write/Read  
TitleState Property**

**About Titles**

<b>Description</b> <b>VB Syntax</b>	Turns ON and OFF the window title. Write a window title with Title <i>win.TitleState = state</i>
<b>Variable</b> <i>win</i> <i>state</i>	<b>(Type) - Description</b> A NaWindow ( <b>object</b> ) <b>(boolean)</b> <b>True (1)</b> - Title ON <b>False (0)</b> - Title OFF

<b>Return Type</b>	Long Integer 0 - Title OFF 1 - Title ON
<b>Default</b>	0 - OFF
<b>Examples</b>	win.TitleState = True 'Write titlestate = win.TitleState 'Read
<b>C++ Syntax</b>	HRESULT get_TitleState(VARIANT_BOOL* bState) HRESULT put_TitleState(VARIANT_BOOL bState)
<b>Interface</b>	INAWindow

### Read-only WindowNumber Property

<b>Description</b>	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.
<b>VB Syntax</b>	<i>value</i> = win.WindowNumber
<b>Variable</b> <i>win</i> <i>value</i>	<b>(Type) - Description</b> A NAWindow (object) <b>(long integer)</b> - Variable to store the returned window number
<b>Return Type</b> <b>Default</b>	Long Integer Not Applicable
<b>Examples</b>	value = app.ActiveNAWindow.WindowNumber
<b>C++ Syntax</b> <b>Interface</b>	HRESULT (long* windowNumber); INAWindow

### Write/Read WindowState Property

### About Arranging Windows

<b>Description</b>	Sets or returns the window setting of Maximized, Minimized, or Normal. To arrange all of the windows, use app.ArrangeWindows.
<b>VB Syntax</b>	<i>object</i> .WindowState = <i>value</i>
<b>Variable</b> <i>object</i>  <i>value</i>	<b>(Type) - Description</b> An Application ( <b>object</b> ) - main window <b>or</b> A NaWindow ( <b>object</b> ) - data windows <b>(enum NAWindowStates)</b> - The window state. Choose from: 0 - <b>naMinimized</b> - Minimizes the window to an Icon on the lower toolbar 1 - <b>naMaximized</b> - Maximizes the window 2 - <b>naNormal</b> - changes the window size to the user defined setting (between Max and Min).
<b>Return Type</b>	Long Integer

<b>Default</b>	naMaximized
<b>Examples</b>	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
<b>C++ Syntax</b>	HRESULT get_WindowState(tagNAWindowStates *pVal) HRESULT put_WindowState(tagNAWindowStates newVal)
<b>Interface</b>	INAWindow IApplication

## Port Extension Object

### Port Extensions Object

#### 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.
Port 1	Sets the Port 1 extension value.
Port 2	Sets the Port 2 extension value.
State	Turns Port Extensions ON and OFF.



#### Write/Read InputA Property

#### About Port Extensions

<b>Description</b>	Sets a Port Extension value for Receiver A
<b>VB Syntax</b>	<i>portExt.InputA = value</i>
<b>Variable</b> <i>portExt</i> <i>value</i>	<b>(Type) - Description</b> A Port Extension <b>(object)</b> <b>(double)</b> - Port Extension value in seconds. Choose any number between <b>-10</b> and <b>10</b>
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	portExt.InputA = 10e-6 'Write inA = portExt.InputA 'Read
<b>C++ Syntax</b>	HRESULT get_InputA(double *pVal) HRESULT put_InputA(double newVal)



**Interface** IPortExtension

**Write/Read  
InputB Property**

**About Port Extensions**

---

<b>Description</b> <b>VB Syntax</b>	Sets the Port Extension value for Receiver B <i>portExt.InputB = value</i>
<b>Variable</b> <i>portExt</i> <i>value</i>	<b>(Type) - Description</b> A Port Extension ( <b>object</b> ) <b>(double)</b> - Port Extension value in seconds. Choose any number between <b>-10</b> and <b>10</b>
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	portExt.InputB = 10e-6 'Write inB = portExt.InputB 'Read
<b>C++ Syntax</b>	HRESULT get_InputB(double *pVal) HRESULT put_InputB(double newVal)
<b>Interface</b>	IPortExtension

**Write/Read  
Port1 Property**

**About Port Extensions**

---

<b>Description</b> <b>VB Syntax</b>	Sets a Port Extension value for Port 1 <i>portExt.Port1 = value</i>
<b>Variable</b> <i>portExt</i> <i>value</i>	<b>(Type) - Description</b> A Port Extension ( <b>object</b> ) <b>(double)</b> - Port Extension value in seconds. Choose any number between <b>-10</b> and <b>10</b>
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	portExt.Port1 = 10e-6 'Write prt1 = portExt.Port1 'Read
<b>C++ Syntax</b>	HRESULT get_Port1(double *pVal) HRESULT put_Port1(double newVal)
<b>Interface</b>	IPortExtension

**Write/Read  
Port2 Property**

**About Port Extensions**

---

<b>Description</b>	Sets a Port Extension value for Port 2
--------------------	--

<b>VB Syntax</b>	<code>portExt.Port2 = value</code>
<b>Variable</b> <code>portExt</code> <code>value</code>	<b>(Type) - Description</b> A Port Extension ( <b>object</b> ) <b>(double)</b> - Port Extension value in seconds. Choose any number between <b>-10</b> and <b>10</b>
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	<code>portExt.Port2 = 10e-6 'Write</code> <code>prt2 = portExt.Port2 'Read</code>
<b>C++ Syntax</b>	HRESULT get_Port2(double *pVal) HRESULT put_Port2(double newVal)
<b>Interface</b>	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

<b>Description</b>	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 <code>seg.Remove</code> .
--------------------	---

<b>VB Syntax</b>	<code>segs.Add (item [ size])</code>
<b>Variable</b> <code>segs</code>	<b>(Type) - Description</b> <b>(object)</b> - A PowerLossSegments collection (object)

<i>item</i>	<b>(variant)</b> - Number of the new segment. If it already exists, a new segment is inserted at the requested position.
<i>size</i>	<b>(long integer)</b> - Optional argument. The number of segments to add, starting with item. If unspecified, value is set to 1.
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	segs.Add 1, 4 'Adds segments 1,2,3 and 4
<hr/>	
<b>C++ Syntax Interface</b>	HRESULT Add(VARIANT index, long size); 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

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 Frequency Property

#### About Source Power Cal

<b>Description</b>	Sets or returns the frequency associated with a PowerSensorCalFactorSegment or Sets or returns the frequency associated with a PowerLossSegment.
<b>VB Syntax</b>	<i>object.Frequency = value</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>object</i>	<b>(object)</b> - PowerSensorCalFactorSegment or PowerLossSegment
<i>value</i>	<b>(double)</b> – Frequency in units of Hz. This can be any non-negative value

<b>Return Type</b>	(limited by the maximum value of double).
<b>Default</b>	Double 0
<b>Examples</b>	seg.Frequency = 6e9 'Write freq = seg.Frequency 'Read
<b>C++ Syntax</b>	HRESULT put_Frequency(double newVal); HRESULT get_Frequency(double *pVal);
<b>Interface</b>	IPowerSensorCalFactorSegment IPowerLossSegment

### Write / Read Loss (Source Power Cal) Property

### About Source Power Cal

<b>Description</b>	Sets or returns the loss value associated with a PowerLossSegment.
<b>VB Syntax</b>	<i>lossSeg.Loss = value</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>lossSeg</i>	<b>(object)</b> - PowerLossSegment
<i>value</i>	<b>(double)</b> – Loss value in dB. This can be any value between 0 and 200.
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	lossSeg.Loss = 0.5 'Write lossVal = lossSeg.Loss 'Read
<b>C++ Syntax</b>	HRESULT put_Loss(Double newVal); HRESULT get_Loss(Double *pVal);
<b>Interface</b>	IPowerLossSegment

### Read-only SegmentNumber Property

### About Segment Sweep

<b>Description</b>	Returns the number of the current segment, PowerSensorCalFactorSegment or PowerLossSegment object.
<b>VB Syntax</b>	<i>seg.SegmentNumber</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>seg</i>	<b>(object)</b> - A Segment, PowerSensorCalFactorSegment or PowerLossSegment. Get a handle to the object by referring to the item in the appropriate collection (Segments, CalFactorSegments or PowerLossSegments).
<b>Return Type</b>	Long Integer
<b>Default</b>	Not Applicable
<b>Examples</b>	segNum = seg.SegmentNumber 'returns the segment number -Read
<b>C++ Syntax</b>	HRESULT get_SegmentNumber(long *pVal)

<b>Interface</b>	ISegment IPowerSensorCalFactorSegment IPowerLossSegment
------------------	---

## PowerSensor Object

### 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

---

<b>Description</b>	Maximum usable frequency specified for the power sensor.
<b>VB Syntax</b>	<i>pwrSensor.MaximumFrequency = value</i>
<b>Variable</b> <i>pwrSensor</i> <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> - A PowerSensor (object) <b>(double)</b> -Frequency in Hertz.
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerSensors(1).MaximumFrequency = 6e9 'Write  MaxFreq = powerCalibrator.PowerSensors(1).MaximumFrequency 'Read
<b>C++ Syntax</b>	HRESULT put_MaximumFrequency(double newVal); HRESULT get_MaximumFrequency(double *pVal);
<b>Interface</b>	IPowerSensor

### Write/Read About Source Power Cal MinimumFrequency (Source Power Cal) Property

---

<b>Description</b>	Minimum usable frequency specified for the power sensor.
<b>VB Syntax</b>	<i>pwrSensor.MinimumFrequency = value</i>
<b>Variable</b> <i>pwrSensor</i> <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> - A PowerSensor (object) <b>(double)</b> -Frequency in Hertz.
<b>Return Type</b>	Double
<b>Default</b>	0
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerSensors(1).MinimumFrequency = 300e3 'Write  MinFreq = powerCalibrator.PowerSensors(1).MinimumFrequency 'Read
<b>C++ Syntax</b>	HRESULT put_MinimumFrequency(double newVal);

**Interface** HRESULT get\_MinimumFrequency(double \*pVal);  
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**

**(object) - A PowerSensor (object)**  
*pwrSensor* NAPowerAcquisitionDevice

**Return Type** Not Applicable

---

**Default**

---

**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

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**                *powerCalibrator*  
*value*

**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

---

Method	Description
Parse	Provides the ability to send a SCPI command from within the COM command.
Properties	
None	

#### Write-Read Parse Method

#### SCPI Command Tree

---

<b>Description</b>	Executes a SCPI command.
<b>VB Syntax</b>	<i>scpi.Parse ("SCPI command")</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>scpi</i>	A ScpiStringParser ( <b>object</b> )
<i>SCPI command</i>	<b>(string)</b> - Any valid SCPI command
<b>Return Type</b>	String
<b>Default</b>	Not Applicable
<b>Examples</b>	<pre>Dim scpi As ScpiStringParser Set scpi = app.ScpiStringParser Dim startfreq As Double startfreq = 100e6</pre>

```

    ,
    scpi.Parse ("Sens:Freq:Start " & startfreq)'Write
    Dim str As String
    str = scpi.Parse ("Sens:Freq:Start?")'Read

```

---

**C++ Syntax  
Interface**

HRESULT Parse(BSTR SCPI\_Command, BSTR \*pQueryResponse)  
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	Enables the IFBandwidth to be set on individual sweep segments.
Option	
Parent	Returns a handle to the current naNetworkAnalyzer application..
Source Power	Enables setting the Source Power for a segment.
Option	



#### Write-only

#### About Segment Sweep

#### Add (segment) Method

---

<b>Description</b>	Adds segments to the Segments collection, but does not turn the segments ON.
<b>VB Syntax</b>	<i>segs.Add (item, [size])</i>
<i>segs</i>	A segments collection ( <b>object</b> )
<i>item</i>	<b>(variant)</b> Number of the new segment. If it already exists, a new segment is inserted at the requested position.
<i>size</i>	<b>(long integer)</b> Optional argument. The number of segments to add, starting with <i>item</i> . If unspecified, value is set to 1.
<b>Return Type</b>	None
<b>Default</b>	None
<b>Examples</b>	Segs.Add 1, 4 'Adds segments 1,2,3,and 4. (does NOT automatically turn segments ON)
<b>C++ Syntax Interface</b>	HRESULT Add(VARIANT index, long size); 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**

<b>Description</b>	Enables the IFBandwidth to be set on individual sweep segments. This property must be set <b>True</b> <b>before</b> <code>seg.IFBandwidth = value</code> is sent. Otherwise, this command will be ignored.
<b>VB Syntax</b>	<code>segs.IFBandwidthOption = value</code>
<b>Variable</b> <i>segs</i> <i>value</i>	<b>(Type) - Description</b> A Segments collection ( <b>object</b> ) <b>(boolean)</b> <b>True</b> - Enables variable IFBandwidth setting for segment sweep <b>False</b> - Disables variable IFBandwidth setting for segment sweep
<b>Return Type</b> <b>Default</b>	Boolean False
<b>Examples</b>	<code>segs.IFBandwidthOption = True 'Write</code> <code>IFOption = IFBandwidthOption 'Read</code>
<b>C++ Syntax</b>	<code>HRESULT get_IFBandwidthOption(VARIANT_BOOL *pVal)</code> <code>HRESULT put_IFBandwidthOption(VARIANT_BOOL newVal)</code>
<b>Interface</b>	ISegments

**Write/Read** **About Source Power**  
**SourcePowerOption Property**

<b>Description</b>	Enables the source power to be set on individual sweep segments. This property must be set <b>True</b> <b>before</b> <code>seg.TestPortPower = value</code> is sent. Otherwise, the test port power command will be ignored.
<b>VB Syntax</b>	<code>segs.SourcePowerOption = state</code>
<b>Variable</b> <i>segs</i> <i>state</i>	<b>(Type) - Description</b> A Segments collection ( <b>object</b> ) <b>(boolean)</b> <b>True (1)</b> - Enables variable TestPortPower to be set segment sweep <b>False (0)</b> - Disables variable TestPortPower to be set segment sweep
<b>Return Type</b> <b>Default</b>	Boolean <b>True</b> - Enabled <b>False</b> - Disabled
<b>Default</b>	False
<b>Examples</b>	<code>segs.SourcePowerOption = True 'Write</code> <code>powerOption = SourcePowerOption 'Read</code>
<b>C++ Syntax</b>	<code>HRESULT get_SourcePowerOption(VARIANT_BOOL *pVal)</code> <code>HRESULT put_SourcePowerOption(VARIANT_BOOL newVal)</code>
<b>Interface</b>	ISegments

## Segment Object

## Segment Object

---

### 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.

---

### Methods

None

Property	Description
centerFrequency	Sets or returns the center frequency of the segment. Shared with the Channel Object
DwellTime	Dwell time value. Shared with the Channel Object
FrequencySpan	Sets or returns the frequency span of the segment. Shared with the Channel Object
IFBandwidth	Sets or returns the IF Bandwidth of the segment. Shared with the Channel Object
NumberOfPoints	Sets or returns the Number of Points of the segment. Shared with the Channel Object
SegmentNumber	Returns the number of the current segment.
StartFrequency	Sets or returns the start frequency of the segment. Shared with the Channel Object
State	Turns On or OFF a segment.
StopFrequency	Sets or returns the stop frequency of the segment. Shared with the Channel Object
TestPortPower	Sets or returns the RF power level of the segment. Shared with the Channel Object



## SourcePowerCalibrator Object

## SourcePowerCalibrator Object

---

### 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.
SetCallInfo	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).
<b>PowerLossSegments (collection)</b>	
PowerMeterGPIBAddress	Specifies the GPIB address of the power meter that will be referenced by this SourcePowerCalibrator object.
<b>PowerSensors (collection)</b>	
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

<b>Description</b>	Aborts a source power cal acquisition sweep that is currently in progress.
<b>VB Syntax</b>	<i>powerCalibrator</i> . <b>AbortPowerAcquisition</b>
<b>Variable</b>	<b>(Type) - Description</b>
<i>powerCalibrator</i>	<b>(object)</b> - A SourcePowerCalibrator object
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>powerCalibrator</i> .AbortPowerAcquisition
<b>C++ Syntax</b>	HRESULT AbortPowerAcquisition();
<b>Interface</b>	ISourcePowerCalibrator

#### Write-only

#### About Source Power Cal

### AcquirePowerReadings Method

<b>Description</b>	Initiates a source power cal acquisition.
<b>VB Syntax</b>	<i>powerCalibrator</i> . <b>AcquirePowerReadings</b> <i>device</i> [, <i>sync</i> ]
<b>Variable</b>	<b>(Type) - Description</b>
<i>powerCalibrator</i>	<b>(object)</b> - A SourcePowerCalibrator object

<i>device</i>	<b>(enum NAPowerAcquisitionDevice)</b> The specific device (sensor on the power meter) to be used for the acquisition. Choose from: <b>0 – naPowerSensor_A</b> <b>1 – naPowerSensor_B</b>
<i>sync</i>	<b>(boolean)</b> Optional argument. If not specified, value is set to False. Choose from: <b>True (1)</b> – The method does not return until this acquisition has completed (the program calling this method is halted while waiting for the method to return). <b>False (0)</b> – 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.
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	powerCalibrator.AcquirePowerReadings naPowerSensor_A, True
<b>C++ Syntax</b>	HRESULT AcquirePowerReadings(tagNAPowerAcquisitionDevice enumAcqDevice, VARIANT_BOOL bSync);
<b>Interface</b>	ISourcePowerCalibrator

**Write-only**  
**ApplyPowerCorrectionValues Method**

**About Source Power Cal**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>powerCalibrator.ApplyPowerCorrectionValues</i>
<b>Variable</b>	<b>(Type) - Description</b>
<i>powerCalibrator</i>	<b>(object)</b> - A SourcePowerCalibrator object
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<b>Examples</b>	powerCalibrator.ApplyPowerCorrectionValues
<b>C++ Syntax</b>	HRESULT ApplyPowerCorrectionValues();
<b>Interface</b>	ISourcePowerCalibrator

**Write-only**  
**SetCallInfo Method (for source power cal)**

**About Source Power Cal**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>powerCalibrator.SetCallInfo calMethod, channel, sourcePort, calPower</i>

<b>Variable</b>	<b>(Type) - Description</b>
<i>powerCalibrator</i>	<b>(object)</b> - A SourcePowerCalibrator object
<i>calMethod</i>	<b>(enum NASourcePowerCalMethod)</b> The method of gathering the source power correction data.
<i>channel</i>	<b>0 – naPowerMeter</b> (the only method currently supported) <b>(long integer)</b> - 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.
<i>sourcePort</i>	<b>(long integer)</b> - Port number on which the source power cal will be performed.
<i>calPower</i>	<b>(double)</b> - Specifies the power level that is expected at the desired reference plane (input or output of DUT) following the source power cal.
<b>Return Type</b>	None
<b>Default</b>	Not Applicable
<hr/>	
<b>Examples</b>	powerCalibrator.SetCalInfo naPowerMeter, 1, 1, -10
<hr/>	
<b>C++ Syntax</b>	HRESULT SetCalInfo(tagNASourcePowerCalMethod calMethod, long channel, long sourcePort, double calPower);
<b>Interface</b>	ISourcePowerCalibrator

**Read-only  
CalPower Property**

**About Source Power Cal**

<b>Description</b>	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.
<b>VB Syntax</b>	<i>value = powerCalibrator.CalPower (chan, sourcePort)</i>
<hr/>	
<b>Variable</b>	<b>(Type) - Description</b>
<i>value</i>	<b>(double)</b> - Variable to store the returned Cal power value in dBm.
<i>powerCalibrator</i>	<b>(object)</b> - A SourcePowerCalibrator object
<i>chan</i>	<b>(long integer)</b> - Channel number of the PNA.
<i>sourcePort</i>	<b>(long integer)</b> - Source port number
<b>Return Type</b>	None
<b>Default</b>	0
<hr/>	
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.CalPower = -10 'Write  power = powerCalibrator.CalPower 'Read
<hr/>	
<b>C++ Syntax</b>	HRESULT get_CalPower(long channel, long sourcePort, double *pVal);
<b>Interface</b>	ISourcePowerCalibrator

**Write / Read  
PowerMeterGPIBAddress Property**

**About Source Power Cal**

<b>Description</b>	Specifies the GPIB address of the power meter that will be referenced by the SourcePowerCalibrator object.
<b>VB Syntax Variable</b> <i>powerCalibrator value</i>	<i>powerCalibrator.PowerMeterGPIBAddress = value</i>
<b>Return Type Default</b>	<b>(Type) - Description</b> <b>(object)</b> - A SourcePowerCalibrator (object) <b>(long integer)</b> – Power meter GPIB address. Choose any number between 0 and 30. Long integer 13
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.PowerMeterGPIBAddress = 13 'Write  pwrMtrAddress = powerCalibrator.PowerMeterGPIBAddress 'Read
<b>C++ Syntax</b>	HRESULT put_PowerMeterGPIBAddress(long newVal); HRESULT get_PowerMeterGPIBAddress(long *pVal);
<b>Interface</b>	ISourcePowerCalibrator

**Write / Read**  
**ReadingsPerPoint Property**

**About Source Power Cal**

<b>Description</b>	For purpose of averaging during source power cal, specifies how many power readings are taken at each frequency point (Averaging factor).
<b>VB Syntax Variable</b> <i>pwrCal value</i>	<i>pwrCal.ReadingsPerPoint = value</i>
<b>Return Type Default</b>	<b>(Type) - Description</b> <b>(object)</b> - A SourcePowerCalibrator (object) <b>(long integer)</b> – Number of power readings. Choose any number between 1 and 100. Long Integer 1
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.ReadingsPerPoint = 2 'Write  numReadings = powerCalibrator.ReadingsPerPoint 'Read
<b>C++ Syntax</b>	HRESULT put_ReadingsPerPoint(long newVal); HRESULT get_ReadingsPerPoint(long *pVal);
<b>Interface</b>	ISourcePowerCalibrator

**Write / Read**  
**UsePowerLossSegments Property**

**About Source Power Cal**

<b>Description</b>	Specifies if subsequent calls to the AcquirePowerReadings method will make use of the loss table (PowerLossSegments).
<b>VB Syntax Variable</b>	<i>pwrCal.UsePowerLossSegments = value</i>
<b>Variable</b>	<b>(Type) - Description</b>



<i>pwrCal</i> <i>value</i>	<b>(object)</b> – A SourcePowerCalibrator (object) <b>(boolean)</b> - <b>False (0)</b> – Do not use loss table <b>True (1)</b> – Use loss table
<b>Return Type</b> <b>Default</b>	Boolean False
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.UsePowerLossSegments = 1 'Write lossTableState = powerCalibrator.UsePowerLossSegments 'Read
<b>C++ Syntax</b>	HRESULT put_UsePowerLossSegments(VARIANT_BOOL bState); HRESULT get_UsePowerLossSegments(VARIANT_BOOL *bState);
<b>Interface</b>	ISourcePowerCalibrator

**Write / Read** **About Source Power Cal**  
**UsePowerSensorFrequencyLimits Property**

<b>Description</b>	Specifies if subsequent calls to the AcquirePowerReadings method will observe frequency values of the MinimumFrequency and MaximumFrequency properties.
<b>VB Syntax</b>	<i>pwrCal.UsePowerSensorFrequencyLimits</i> = <i>value</i>
<b>Variable</b> <i>pwrCal</i> <i>value</i>	<b>(Type) - Description</b> <b>(object)</b> – A SourcePowerCalibrator (object) <b>(boolean)</b> - <b>False (0)</b> – Do not use power sensor frequency limits. An acquisition will use just one power sensor for the entire sweep, regardless of frequency. <b>True (1)</b> – 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.
<b>Return Type</b> <b>Default</b>	Boolean False (0)
<b>Examples</b>	Set powerCalibrator = pna.SourcePowerCalibrator powerCalibrator.UsePowerSensorFrequencyLimits = 1 'Write FreqCheck = powerCalibrator.UsePowerSensorFrequencyLimits 'Read
<b>C++ Syntax</b>	HRESULT put_UsePowerSensorFrequencyLimits(VARIANT_BOOL bState); HRESULT get_UsePowerSensorFrequencyLimits(VARIANT_BOOL *bState);
<b>Interface</b>	ISourcePowerCalibrator

## Trace Object

## Trace Object

---

### 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

*trac*

*value*

#### Return Type

#### Default

#### (Type) - Description

A Trace **(object)**

**(String)** Trace name

String

"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

---

<b>Description</b>	Sets or returns the Reference Position of the active trace.
<b>VB Syntax</b>	<i>trce.ReferencePosition</i> = <i>value</i>
<b>Variable</b> <i>trce</i> <i>value</i>	<b>(Type) - Description</b> A Trace ( <b>object</b> ) <b>(double)</b> - Reference position on the screen measured in horizontal graticules from the bottom of the screen. Choose from any number between: <b>0</b> and <b>10</b> .
<b>Return Type</b> <b>Default</b>	Double 0
<b>Examples</b>	meas.ReferencePosition = 5 'Middle of the screen -Write rpos = meas.ReferencePosition -Read
<b>C++ Syntax</b>	HRESULT get_ReferencePosition(double *pVal) HRESULT put_ReferencePosition(double newVal)
<b>Interface</b>	ITrace

## Write/Read ReferenceValue Property

---

### About Reference Level

<b>Description</b>	Sets or returns the value of the Y-axis Reference Level of the active trace.
<b>VB Syntax</b>	<i>trce.ReferenceValue</i> = <i>value</i>
<b>Variable</b> <i>trce</i> <i>value</i>	<b>(Type) - Description</b> A Trace ( <b>object</b> ) <b>(double)</b> - Reference Value. Units and range depend on the current data format.
<b>Return Type</b> <b>Default</b>	Double Not applicable
<b>Examples</b>	meas.ReferenceValue = 0 'Write rlev = meas.ReferenceValue 'Read
<b>C++ Syntax</b>	HRESULT get_ReferenceValue(double *pVal) HRESULT put_ReferenceValue(double newVal)
<b>Interface</b>	ITrace

## Write/Read YScale Property

---

### About Scale

<b>Description</b>	Sets or returns the Y-axis Per-Division value of the active trace.
<b>VB Syntax</b>	<i>trace.YScale</i> = <i>value</i>
<b>Variable</b> <i>trace</i>	<b>(Type) - Description</b> A Trace ( <b>object</b> )

<i>value</i>	<b>(double)</b> - Scale /division number. Units and range depend on the current data format.
<b>Return Type</b>	Double
<b>Default</b>	10 (db)
<b>Examples</b>	trac.YScale = 5 'Write yscl = trac.YScale 'Read
<b>C++ Syntax</b>	HRESULT get_YScale(double *pVal) HRESULT put_YScale(double newVal)
<b>Interface</b>	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

---

#### 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**

---

<b>Description</b> VB Syntax	Set the start frequencies when <b>trans.Mode = LowPass</b> . <i>trans.SetFrequencyLowPass</i>
<b>Variable</b> <i>trans</i>	<b>(Type) - Description</b> A Transform ( <b>object</b> )
<b>Return Type</b>	Not Applicable
<b>Default</b>	Not Applicable
<b>Examples</b>	<i>trans.SetFrequencyLowPass</i>
<b>C++ Syntax</b> Interface	HRESULT SetFrequencyLowPass(void) ITransform

**Write/Read** **About Time Domain**  
**ImpulseWidth Property**

---

<b>Description</b> VB Syntax	Sets or returns the Impulse Width of Time Domain transform windows <i>trans.ImpulseWidth = value</i>
<b>Variable</b> <i>trans</i> <i>value</i>	<b>(Type) - Description</b> A Transform ( <b>object</b> ) <b>(double)</b> - Impulse Width in seconds. Range of settings depends on the frequency range of your analyzer.
<b>Return Type</b>	Double
<b>Default</b>	.98 / Default Span
<b>Examples</b>	<i>trans.ImpulseWidth = 200e-12</i> 'sets the Impulse width of a transform window - Write <i>IW = trans.ImpulseWidth</i> 'Read
<b>C++ Syntax</b> Interface	HRESULT get_ImpulseWidth(double *pVal) HRESULT put_ImpulseWidth(double newVal) ITransform

**Write/Read** **About Time Domain**  
**KaiserBeta Property**

---

<b>Description</b> <b>VB Syntax</b>	Sets or returns the Kaiser Beta of Time Domain transform windows <i>trans.KaiserBeta = value</i>
<b>Variable</b> <i>trans</i> <i>value</i>	<b>(Type) - Description</b> A Transform ( <b>object</b> ) <b>(single)</b> - Kaiser Beta. Choose any number between <b>0</b> and <b>13</b> .
<b>Return Type</b> <b>Default</b>	Single 0
<b>Examples</b>	<i>trans.KaiserBeta = 6</i> 'sets the Kaiser Beta of a transform window -Write <i>KB = trans.KaiserBeta</i> 'Read
<b>C++ Syntax</b>	HRESULT get_KaiserBeta(float *pVal) HRESULT put_KaiserBeta(float newVal)
<b>Interface</b>	ITransform

**Write/Read**  
**Mode Property**

**About Time Domain**

<b>Description</b> <b>VB Syntax</b>	Sets the type of transform. <i>trans.Mode = value</i>
<b>Variable</b> <i>trans</i> <i>value</i>	<b>(Type) - Description</b> A Transform ( <b>object</b> ) <b>(enum NATransformMode)</b> - Choose from: <b>0</b> - naTransformBandpassImpulse <b>1</b> - naTransformLowpassImpulse <b>2</b> - naTransformLowpassStep
<b>Return Type</b> <b>Default</b>	NATransformMode <b>0</b> - naTransformBandpassImpulse
<b>Examples</b>	<i>trans.Mode = naTransformLowpassStep</i> 'Write <i>transmode = trans.Mode</i> 'Read
<b>C++ Syntax</b>	HRESULT get_Mode(tagNATransformMode *pVal) HRESULT put_Mode(tagNATransformMode newVal)
<b>Interface</b>	ITransform

**Write/Read**  
**StepRiseTime Property**

**About Time Domain**

<b>Description</b> <b>VB Syntax</b>	Sets or returns the Rise time of the stimulus in Low Pass Step Mode. <i>trans.StepRiseTime = value</i>
<b>Variable</b> <i>trans</i> <i>value</i>	<b>(Type) - Description</b> A Transform ( <b>object</b> ) <b>(double)</b> - Rise time in seconds. Choose any number between <b>5.0e-13</b> and <b>1.63e-14</b> .
<b>Return Type</b>	Double

<b>Default</b>	0
<b>Examples</b>	trans.StepRiseTime = 1.0e-14 'sets the step rise time to 100 psec. -Write rt = trans.StepRiseTime 'Read
<b>C++ Syntax</b>	HRESULT get_StepRiseTime(double *pVal) HRESULT put_StepRiseTime(double newVal)
<b>Interface</b>	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](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

```

---

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

```

---

Intro to Examples

---

## 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

```

```

    limits(i).Type = naLimitSegmentType_OFF
Next i

'Set up Limit Lines
limits(1).Type = naLimitSegmentType_Maximum
limits(1).BeginResponse = 10
limits(1).EndResponse = 10
limits(1).BeginStimulus = 4000000000#
limits(1).EndStimulus = 8000000000#
limits(2).Type = naLimitSegmentType_Minimum
limits(2).BeginResponse = -30
limits(2).EndResponse = -30
limits(2).BeginStimulus = 4000000000#
limits(2).EndStimulus = 8000000000#

'Turn on Lines, Testing, and Sound
limits.LineDisplay = 1
limits.State = 1
limits.SoundOnFail = 1

```

---

## Intro to Examples

---

### 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

```

---

## Intro to Examples

---

### 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

```

---



---

## Intro to Examples

---

### 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

```

---

## Intro to Examples

### 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;
}

```

---



---

## Intro to Examples

---

### 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

```

---



---

## Intro to Examples

---

### 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](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
- Read and Write Calibration Data using COM
- C++ and the COM Interface

### 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 Analyzer on Your PC
- Problems?

---

**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 **regsvr32 "C:\program files\common files\agilent\pna\835xps.dll"**

After doing these, perform "Access Procedure" (run dcomcnfg).



### 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.

### 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

**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 are 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



**CoInitializeEx( )**. Alternatively you can use the MFC (Microsoft Foundation Classes) **AfxOleInit( )**.

Conversely, before your program exits you must uninitialized 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  
  CLSCTX_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 lunkown 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.



---

## 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)
```

## 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

---

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 interrupt 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.

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

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
Se	C	R	Facility													Code															
v																															

## 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	<code>naEventSeveritySUCCESS</code> - the operation completed successfully
01	<code>naEventSeverityINFORMATIONAL</code> - events that occur without impact on the measurement integrity
10	<code>naEventSeverityWARNING</code> - events that occur with potential impact on measurement integrity
11	<code>naEventSeverityERROR</code> - 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
<code>naEventCategory_PARSER</code>	<code>OnSCPIEvent</code>
<code>naEventCategory_MEASURE</code>	<code>OnMeasurementEvent</code>
<code>naEventCategory_CHANNEL</code>	<code>OnChannelEvent</code>
<code>naEventCategory_HW</code>	<code>OnHardwareEvent</code>
<code>naEventCategory_CAL</code>	<code>OnCalEvent</code>
<code>naEventCategory_USER</code>	<code>OnUserEvent</code>
<code>naEventCategory_DISPLAY</code>	<code>OnDisplayEvent</code>
<code>naEventCategory_GENERAL</code>	<code>OnSystemEvent</code>

**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 explicitly.

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

<b>ABORt</b>	Stops all sweeps
 <b>CALCulate</b>	Click to hide CALC commands
<b>:CORRection</b>	Sets Electrical Delay and Phase Offset
<b>:DATA</b>	Sends and queries data.
<b>:FILTer</b>	Sets time domain gating
<b>:FORMat</b>	Sets the display format
<b>:FUNCTioN</b>	Controls Trace Statistics
<b>:LIMit</b>	Controls limit lines for pass / fail testing
<b>:MARKer</b>	Controls the marker settings
<b>:MATH</b>	Performs math on the memory trace
<b>:NORMalize</b>	Specifies the normalization features used for a receiver power calibration
<b>:PARAmeter</b>	Creates and deletes measurements
<b>:RDATa?</b>	Queries receiver data
<b>:SMOothing</b>	Controls point-to-point smoothing
<b>:TRANSform</b>	Controls time domain transform settings
<b>CONTRol</b>	Controls the rear-panel connectors
<b>DISPlay</b>	Controls the display settings
<b>FORMat</b>	Sets the format for data transfer
<b>HCOPY</b>	Controls hardcopy printing
<b>INITiate</b>	Sets continuous or manual triggering

**MMEMory**  
**OUTPut**

Saves and recalls instrument states  
Turns RF power ON and OFF

 **SENSE** Click to hide SENSE commands

**:AVERage**  
**:BANDwidth**  
**:CORRection**  
**:CORR:COLL:CKIT**  
**:CORR:CSET**  
**:CORR:GUIDed**  
**:COUPle**  
**:FREQuency**  
**:POWer**

Sets sweep averaging parameters  
Specifies the IF filter bandwidth  
Provides non-guided calibration capability  
Defines calibration standards  
Manages Cal Sets  
Provides Guided calibration capability  
Sets sweep as Chopped or Alternate  
Controls frequency sweep functions  
Sets receiver attenuation and overpower protection  
Returns the source of the reference oscillator.  
Defines the segment sweep settings.  
Specifies the sweep modes of the analyzer.  
Controls the power to the DUT  
Provides for Source Power Correction  
Reads the analyzer status registers  
Controls the analyzer defaults  
Starts or ends a measurement

**:ROSCillator**  
**:SEGment**  
**:SWEep**  
**SOURce**  
**SOURce:POWer**  
**STATus**  
**SYSTem**  
**TRIGger**



## 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 exist.

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 INITiate: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

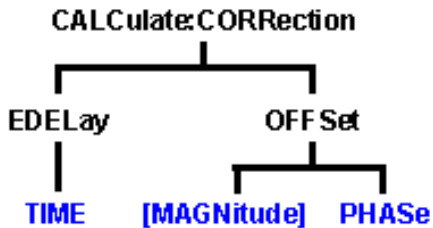
Default Not applicable

---

## 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: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 Return Type**

```
CALCulate:CORRection:EDELay:TIME?
Character
```

#### **Overlapped? Default**

```
No
0 seconds
```

---

### **CALCulate<cnum>:CORRection:OFFSet[:MAGNitude] <num>**

(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. **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.

<b>Examples</b>	<code>CALC:CORR:OFFS -10DBM</code> <code>calculate1:correction:offset:magnitude maximum</code>
<b>Query Syntax</b> <b>Return Type</b>	<code>CALCulate&lt;cnum&gt;:CORRection:OFFSet[:MAGNitude]?</code> Character
<b>Overlapped?</b> <b>Default</b>	No 0dBm

### **CALCulate<cnum>:CORRection:OFFSet:PHASe <num>[<char>]**

(Read-Write) Sets the phase offset for the selected measurement. **Critical Note:**

#### **Parameters**

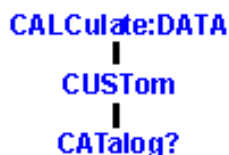
<code>&lt;cnum&gt;</code>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <code>&lt;cnum&gt;</code> is set to 1.
<code>&lt;num&gt;</code>	Offset phase value. Choose any number between: <b>-360</b> and <b>360</b>
<code>&lt;char&gt;</code>	Units for phase. OPTIONAL. Choose either: <b>DEG</b> - Degrees (default) <b>RAD</b> - Radians

<b>Examples</b>	<code>CALC:CORR:OFFS:PHAS 10</code> <code>calculate:correction:offset:phase 20rad</code>
<b>Query Syntax</b> <b>Return Type</b>	<code>CALCulate:CORRection:OFFSet:PHASe?</code> Character, returned value always in degrees
<b>Overlapped?</b> <b>Default</b>	No 0 degrees



## **Calc:Data Commands**

Controls sending and receiving data with the PNA



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- See Data Access Map

## **CALCulate<num>:DATA <char>,<data>**

Writes Measurement data, Memory data, Normalization Divisor data, or Error terms.

---

## **CALCulate<num>: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**

<num> - Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> 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...
<b>SCORR3</b>	Reflection Tracking

For **Response Short** calibrations:

Specify this <char>...	to get this Term...
<b>SCORR3</b>	Reflection Tracking

For **Response Thru** calibrations:

Specify this <char>...	to get this Term...
<b>SCORR6</b>	Transmission Tracking

For **Response Thru and Isolation** calibrations:

Specify this <char>...	to get this Term...
<b>SCORR4</b>	Isolation
<b>SCORR6</b>	Transmission Tracking

For **1-Port** calibrations:

Specify this <char>...	to get this Term...
<b>SCORR1</b>	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 Return Type**

CALCulate:DATA:CUSTom? <name>  
**REAL or ASCII** (see Getting Data from the Analyzer)

---

#### **Overlapped? Default**

No  
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? Default**

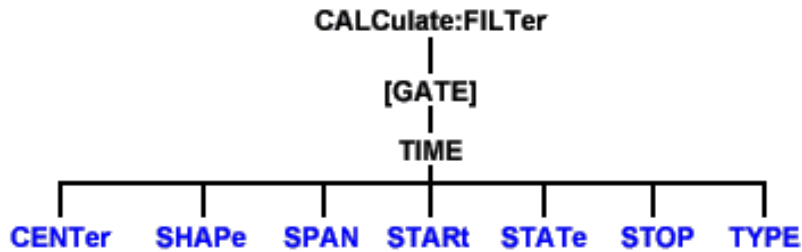
No  
Not Applicable



## **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:  
 $\pm (\text{number of points}-1) / \text{frequency span}$
- 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 Return Type

```
CALCulate<cnUm>:FILTer[:GATE]:TIME:CENTer?
Character
```

#### Overlapped? Default

```
No
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?
```



<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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 $2^* [(number\ of\ points-1) / frequency\ span]$ <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> 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 Return Type**

```
CALCulate<cnum>:FILTer[:GATE]:TIME:SPAN?
Character
```

#### **Overlapped? Default**

```
No
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>	<b>ON</b> (or 1) - turns gating ON. <b>OFF</b> (or 0) - turns gating OFF.

#### **Examples**

```
CALC:FILT:TIME:STAT ON
calculate2:filter:gate:time:state off
```

#### **Query Syntax Return Type**

```
CALCulate<cnum>:FILTer[:GATE]:TIME:STATE?
Boolean (1 = ON, 0 = OFF)
```

#### **Overlapped? Default**

```
No
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: $\pm (number\ of\ points-1) / frequency\ span$ <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> instead of a numeric parameter. See SCPI Syntax for more information.

<b>Examples</b>	CALC:FILT:TIME:STAR 1e-8 calculate2:filter:gate:time:start minimum
<b>Query Syntax</b>	CALCulate<cnum>:FILTer[:GATE]:TIME:STARt?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	CALC:FILT:TIME:STOP -1 ns calculate2:filter:gate:time:stop maximum
<b>Query Syntax</b>	CALCulate<cnum>:FILTer[:GATE]:TIME:STOP?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	CALC:FILT:TIME BPAS calculate2:filter:gate:time:type notch
<b>Query Syntax</b>	CALCulate<cnum>:FILTer[:GATE]:TIME[:TYPE]?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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 Return Type

```
CALCulate<cnum>:FORMat?  
Character
```

---

#### Overlapped? Default

```
No  
MLINear
```

---

List of all commands in this block:

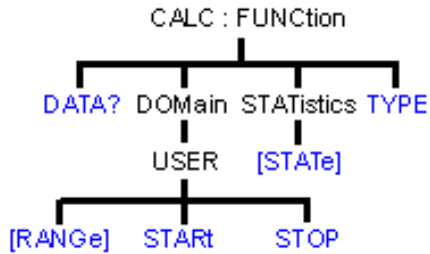
(Parameters in ***bold italics***)

```
: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**

<cnnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnnum> 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  
Return Type**

CALCulate<cnnum>:FUNCtion:DOMain:USER:STARt? <range>  
Character

**Overlapped?  
Default**

No  
The analyzer's **Minimum** x-axis value

**CALCulate<cnnum>: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**

<cnnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnnum> 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  
Return Type**

CALCulate<cnnum>:FUNCtion:DOMain:USER:STOP? <range>  
Character

**Overlapped?  
Default**

No  
The analyzer's **Maximum** x-axis value

**CALCulate<cnnum>:FUNCtion:STATistics[:STATe] <ONIOFF>**

(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**

<cnnum> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnnum> is set to 1.  
 <ONIOFF> ON - Displays trace statistics

	OFF - Hides trace statistics
<b>Examples</b>	<code>CALC:FUNC:STAT ON</code> <code>calculate2:function:statistics:state off</code>
<b>Query Syntax</b> <b>Return Type</b>	<code>CALCulate&lt;cnum&gt;:FUNCTION:STATistics[:STATe]?</code> Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b> <b>Default</b>	No OFF (0)

### **CALCulate<cnum>:FUNCTION:TYPE <char>**

(Read-Write) Sets statistic TYPE that you can then query using `CALC:FUNCTION:DATA?`.

**Critical Note:**

**Parameters**

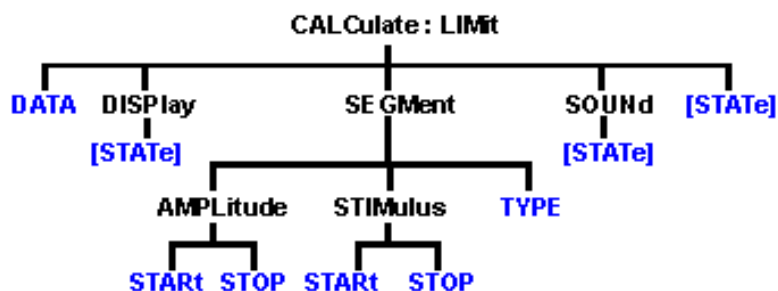
<cnum>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<char>	Choose from: <b>PTPeak</b> - the difference between the max and min data points on the trace. <b>STDEV</b> - standard deviation of all data points on the trace <b>MEAN</b> - mean (average) of all data points on the trace

<b>Examples</b>	<code>CALC:FUNC:TYPE PTP</code> <code>calculate2:function:type stdev</code>
<b>Query Syntax</b> <b>Return Type</b>	<code>CALCulate&lt;cnum&gt;:FUNCTION:TYPE?</code> Character
<b>Overlapped?</b> <b>Default</b>	No 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

<b>&lt;cnum&gt;</b>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<b>&lt;block&gt;</b>	Data for all limit segments in REAL,64 format. The following is the data format for 1 segment: <b>Type,BegStim, EndStim, BegResp,EndResp</b> <b>Type</b> Type of limit segment. Choose from 0 - Off 1 - Max 2 - Min  <b>BegStim</b> Start of X-axis value (freq, power, time)  <b>EndStim</b> End of X-axis value  <b>BegResp</b> Y-axis value that corresponds with Start of X-axis value  <b>EndResp</b> 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 Return Type

CALCulate<cnum>:LIMit:DATA?  
Definite length block - All 100 predefined limit segments are returned.

---

#### Overlapped? Default

No  
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

<b>&lt;cnum&gt;</b>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnum> is set to 1.
<b>&lt;ON   OFF&gt;</b>	<b>ON</b> (or 1) - turns the display of limit segments ON. <b>OFF</b> (or 0) - turns the display of limit segments OFF.

<b>Examples</b>	CALC:LIM:DISP:STAT ON calculate2:limit:display:state off
<b>Query Syntax Return Type</b>	CALCulate<cnum>:LIMit:DISPlay[:STATe]? Boolean (1 = ON, 0 = OFF)
<b>Overlapped? Default</b>	No 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.

<b>Examples</b>	CALC:LIM:SEGM1:AMPL:STAR 10 calculate2:limit:segment2:amplitude:start 10
<b>Query Syntax Return Type</b>	CALCulate<cnum>:LIMit:SEGment<snum>AMPLitude:STARt? Character
<b>Overlapped? Default</b>	No 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.

<b>Examples</b>	CALC:LIM:SEGM1:AMPL:STOP 10 calculate2:limit:segment2:amplitude:stop 10
<b>Query Syntax Return Type</b>	CALCulate<cnum>:LIMit:SEGment<snum>AMPLitude:STOP? Character
<b>Overlapped? Default</b>	No 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  
Return Type**

CALCulate<cnum>:LIMit:SEGment<snum>STIMulus:STARt?  
Character

---

**Overlapped?  
Default**

No  
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  
Return Type**

CALCulate<cnum>:LIMit:SEGment<snum>STIMulus:STOP?  
Character

---

**Overlapped?  
Default**

No  
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: <b>1 and 100</b> If unspecified, value is set to 1.
<char>	Choose from: <b>LMAX</b> - a MAX limit segment. Any response data exceeding the MAX value will fail. <b>LMIN</b> - a MIN limit segment. Any response data below the MIN value will fail. <b>OFF</b> - the limit segment (display and testing) is turned OFF.

---

**Examples**

```
CALC:LIM:SEGM:TYPE LMIN
calculate2:limit:segment3:type lmax
```

---

**Query Syntax  
Return Type**

CALCulate<cnum>:LIMit:SEGment<snum>:TYPE?  
Character

---

**Overlapped?  
Default**

No  
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 Return Type**

CALCulate<cnum>:LIMit:SOUNd[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

#### **Overlapped? Default**

No  
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 Return Type**

CALCulate<cnum>:LIMit:STATe?  
Boolean (1 = ON, 0 = OFF)

---

#### **Overlapped? Default**

No  
OFF

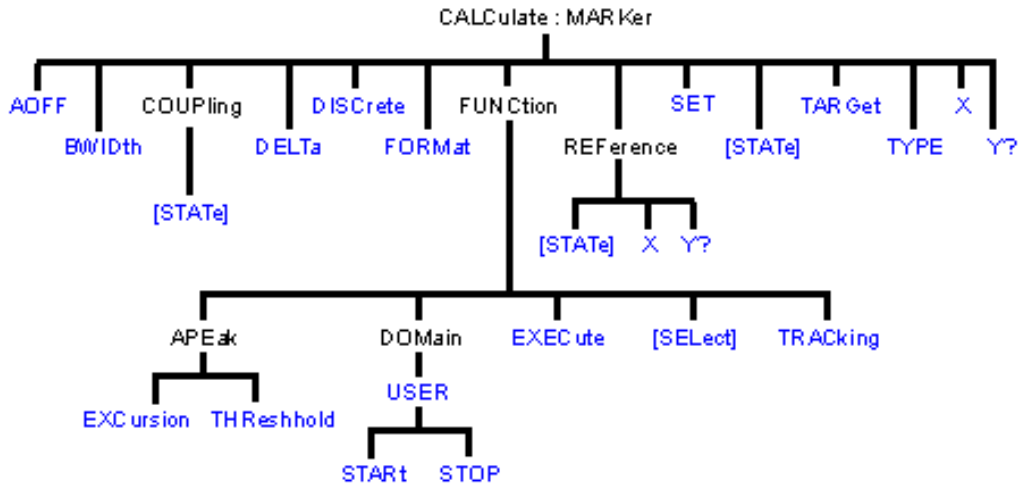
---



## **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**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.  
<num> Target value below filter peak. Choose any number between: **-500 and 500**

**Examples**

```
CALC:MARK:BWID -3  
calculate2:marker:ewidth -2.513
```

**Query Syntax**

CALCulate<num>:MARKer:BWIDth?  
Returns the results of bandwidth search:

**Return Type**

Four Character values separated by commas: bandwidth, center Frequency, Q, loss.

**Overlapped?**

No

**Default**

-3

**CALCulate<num>:MARKer<mrk>:COUPling[:STATe]<ONIOFF>**

(Read-Write) Sets and Reads the state of Coupled Markers (ON and OFF) **Critical Note:**

**Parameters**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.  
<mrk> Any existing marker number from 1 to 10; if unspecified, value is set to 1.  
<ONIOFF> **False (0)** - Turns Coupled Markers OFF  
**True (1)** - Turns Coupled Markers ON

**Examples**

```
CALC:MARK:COUP ON  
calculate2:marker8:coupling off
```

**Query Syntax**

CALCulate<num>:MARKer<mrk>:COUPling:[STATe]?

**Return Type**

Boolean (1 = ON, 0 = OFF)

**Overlapped?**

No

**Default**

OFF

**CALCulate<num>:MARKer<mrk>:DELTA <ONIOFF>**

(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**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.  
<mrk> Any existing marker number from 1 to 10; if unspecified, value is set to 1.  
<ONIOFF> **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<num>:MARKer<mrk>:DELTA?

**Return Type**

Boolean (1 = ON, 0 = OFF)

**Overlapped?**

No

Default OFF

---

### CALCulate<cnum>:MARKer<mkr>:DISCrete <ONIOFF>

(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.  
<ONIOFF> **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 Return Type

CALCulate<cnum>:MARKer<mkr>:DISCrete?  
Boolean (1 = ON, 0 = OFF)

---

#### Overlapped? Default

No  
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)  
**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 Return Type

CALCulate<cnum>:MARKer<mkr>:FORMat?  
Character

---

#### Overlapped? Default

No  
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**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> 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  
Return Type**

CALCulate<num>:MARKer<mkr>:FUNCTion:APEak:EXCursion?  
Character

**Overlapped?  
Default**

No  
3

**CALCulate<num>: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**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> 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  
Return Type**

CALCulate<num>:MARKer<mkr>:FUNCTion:APEak:THReshold?  
Character

**Overlapped?  
Default**

No  
-100

**CALCulate<num>: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**

<num> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> 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
```

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:FUNction:DOMain:USER:STOP?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	The analyzer's <b>MAXimum</b> 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: <b>MAXimum</b> - finds the highest value <b>MINimum</b> - finds the lowest value <b>RPEak</b> - finds the next valid peak to the right <b>LPEak</b> - finds the next valid peak to the left <b>NPEak</b> - finds the <b>next highest</b> value among the valid peaks <b>TARGet</b> - finds the target value to the right, wraps around to the left <b>LTARGet</b> - finds the next target value to the left of the marker <b>RTARGet</b> - finds the next target value to the right of the marker

**Examples**

```
CALC:MARK:FUNC:EXEC
calculate2:marker2:function:execute maximum
```

<b>Query Syntax</b>	Not applicable
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>MAXimum</b> - finds the highest value <b>MINimum</b> - finds the lowest value <b>RPEak</b> - finds the next valid peak to the right <b>LPEak</b> - finds the next valid peak to the left <b>NPEak</b> - finds the <b>next highest</b> value among the valid peaks <b>TARGet</b> - finds the target value to the right; wraps around to the left <b>LTARGet</b> - finds the next target value to the left of the marker <b>RTARGet</b> - finds the next target value to the right of the marker



<b>Examples</b>	CALC:MARK:FUNC MAX calculate2:marker8:function:select ltarget
<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:FUNction[:SElect]?
<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	CALC:MARK:TARG 2.5 calculate2:marker8:target -10.3
-----------------	---

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:TARGet?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (or 1) - The specified marker will "Track" (find) the selected function every sweep. <b>OFF</b> (or 0) - The specified marker will find the selected function <b>only</b> when the CALC:MARK:FUNC:EXECute command is sent.

<b>Examples</b>	CALC:MARK:FUNC:TRAC ON calculate2:marker8:function:tracking off
-----------------	--

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:FUNction:TRACking?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)

<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (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  
Return Type**

CALCulate<cnum>:MARKer:REFerence[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

**Overlapped?  
Default**

No  
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  
Return Type**

CALCulate<cnum>:MARKer:REFerence:X?  
Character

---

**Overlapped?  
Default**

No  
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?  
Default**

No  
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
```

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:TYPE?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>CENTer</b> - changes center frequency to the value of the marker <b>SPAN</b> - changes the frequency span to the value of the marker's domain <b>STARt</b> - changes the start frequency to the value of the marker <b>STOP</b> - changes the stop frequency to the value of the marker <b>RLEVel</b> - changes the reference level to the value of the marker <b>DELay</b> - changes the xxx delay to the value of the marker

**Examples**

```
CALC:MARK:SET CENT
calculate2:marker8:set span
```

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:SET?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	Not applicable

### CALCulate<cnum>:MARKer<mkr>[:STATe] <ONIOFF>

(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.
<ONIOFF>	<b>ON</b> (or 1) - turns marker ON. <b>OFF</b> (or 0) - turns marker OFF.

**Examples**

```
CALC:MARK ON
calculate2:marker8 on
```

<b>Query Syntax</b>	CALCulate<cnum>:MARKer<mkr>:STATe?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	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
--------	---

<mkr> measurement on that channel. If unspecified, <cnum> is set to 1.  
 <num> Any marker number from 1 to 10; if unspecified, value is set to 1.  
 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  
Return Type**

CALCulate<cnum>:MARKer<mkr>:X?  
 Character

**Overlapped?  
Default**

No  
 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  
Return Type**

CALCulate<cnum>:MARKer<mkr>:Y?  
 Character

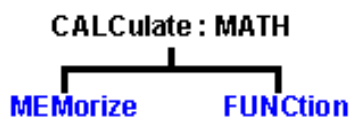
**Overlapped?  
Default**

No  
 Not applicable



**Calc:Math Command**

Controls math operations on the currently selected measurement and memory.



- 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<num>: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

<num>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.
<char>	The math operation to be applied. Choose from the following:
	<b>NORMAL</b> Trace data only
	<b>ADD</b> Data + Memory
	<b>SUBTRACT</b> Data - Memory
	<b>MULTIPLY</b> Data * Memory
	<b>DIVIDE</b> Data / Memory

#### Examples

```
CALC:MATH:FUNC NORM
calculate2:math:function subtract
```

#### Query Syntax Return Type

CALCulate<num>:MATH:FUNCTION?  
Character

#### Overlapped? Default

No  
NORMAL

### CALCulate<num>:MATH:MEMorize

(Write-only) Puts the currently selected measurement trace into memory. (Data-> Memory)

**Critical Note:**

#### Parameters

<num>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.
-------	---

#### Examples

```
CALC:MATH:MEM
calculate2:math:memorize
```

#### Query Syntax

Not applicable

#### Overlapped? Default

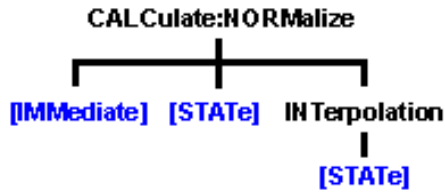
No  
Not applicable



## 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
```

<b>Query Syntax</b>	CALCulate<cnum>:NORMalize:STATe?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	OFF

### CALCulate<cnum>:NORMalize:INTerpolation[: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>	<b>ON (or 1)</b> – turns interpolation ON. <b>OFF (or 0)</b> – turns interpolation OFF.

#### Examples

```
CALC:NORM:INT ON
calculate2:normalize:interpolation:state off
```

#### Query Syntax

CALCulate<cnum>:NORMalize:INTerpolation[:STATe]?  
Boolean (1 = ON, 0 = OFF)

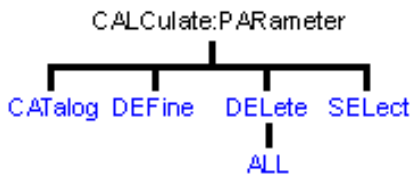
#### Overlapped?

No  
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 Return Type

CALCulate<cnum>:PARAmeter:CATalog?  
String - "<measurement name>,<parameter>,[<measurement name>,<parameter>...]"

### Overlapped? Default

No  
"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 | S22 | S12 | S21**

For 3-port analyzers only:

**S33 | S13 | S31 | S23 | 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
<b>AB</b>	A/B
<b>AC</b>	A/C - 3 port analyzers only
<b>BA</b>	B/A
<b>BC</b>	B/C - 3 port analyzers only
<b>CA</b>	C/A - 3 port analyzers only



<b>CB</b>	C/B - 3 port analyzers only
<b>AR1</b>	A/R1
<b>BR1</b>	B/R1
<b>CR1</b>	C/R1 - 3 port analyzers only
<b>AR2</b>	A/R2
<b>BR2</b>	B/R2
<b>R1A</b>	R1/A
<b>R2A</b>	R2/A
<b>R1B</b>	R1/B
<b>R2B</b>	R2/B
<b>R1C</b>	R1/C - 3 port analyzers only
<b>R2R1</b>	R2/R1
<b>R1R2</b>	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<cnun>:PARAmeter:DELeTe [:NAME]<Mname>**

(Write-only) Deletes the specified measurement. **Critical Note:**

**Parameters**

<cnun> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnun> 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<cnun>:PARAmeter:DELeTe:ALL**

(Write-only) Deletes all specified measurements. **Critical Note:**

**Parameters**

<cnun> Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnun> is set to 1.

**Examples**

```
CALC:PAR:DEL:ALL
  calculate2:parameter:delete:all
```

**Query Syntax**

Not Applicable

<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

### **CALCulate<num>: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**

<num>	Channel number of the measurement to be selected. If unspecified, <num> 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'
```

<b>Query Syntax</b>	CALCulate:PARAmeter:SElect?
<b>Return Type</b>	String

<b>Overlapped?</b>	No
<b>Default</b>	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<num>:RDATA? <char>**

(Read-only) Returns receiver data for the selected measurement. To query measurement data, see CALC:DATA? **Critical Note:**

#### **Parameters**

<num>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <num> is set to 1.
<char>	Choose from receivers: <b>A</b>

	<b>B</b>
	<b>R1</b>
	<b>R2</b>
	<b>REF</b> - returns either R1 or R2 data depending on the source port of the CALC-selected measurement.
<b>Example</b>	GPIB.Write "INITiate:CONTInuous OFF" GPIB.Write "INITiate:IMMediate;*wai" GPIB.Write "CALCulate:RDATA? A"
<b>Return Type</b>	Character - Two numbers per data point
<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

List of all commands in this block:  
 (Parameters in ***bold italics***)

:CALCulate***1***: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<cnm>: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

<cnm>	Channel number of the measurement. There must be a selected measurement on that channel. If unspecified, <cnm> is set to 1.
<num>	Percentage value. Choose any number between: <b>1 and 25</b>

**Examples**      CALC:SMO:APER 2

	calculate2:smoothing:aperture 20.7
<b>Query Syntax</b>	CALCulate<cnum>:SMOothing:APERture?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	CALC:SMO:POIN 50 calculate2:smoothing:points 21
-----------------	--

<b>Query Syntax</b>	CALCulate<cnum>:SMOothing:POINTs?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (or 1) - turns smoothing ON. <b>OFF</b> (or 0) - turns smoothing OFF.

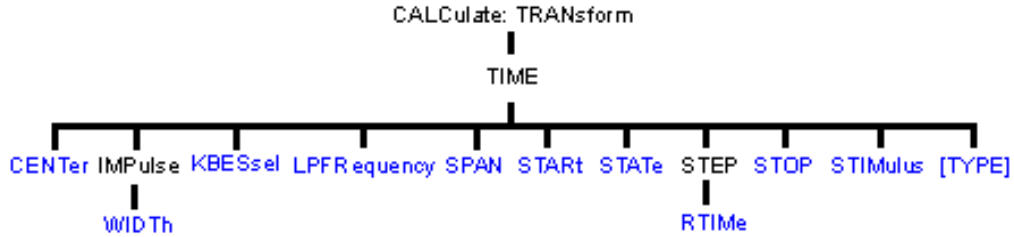
<b>Examples</b>	CALC:SMO ON calculate2:smoothing:state off
-----------------	---

<b>Query Syntax</b>	CALCulate<cnum>:SMOothing[:STATe]?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	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:  
 $\pm (\text{number of points}-1) / \text{frequency span}$
- 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 Return Type

CALCulate<cnum>:TRANSform:TIME:CENTer?  
Character

#### Overlapped? Default

No  
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:  
 $.6 / \text{frequency span}$  and  $1.39 / \text{frequency span}$

#### Examples

```
CALC:TRAN:TIME:IMP:WIDTh 10
calculate2:transform:time:impulse:width 13
```

#### Query Syntax Return Type

CALCulate<cnum>:TRANSform:TIME:IMPulse:WIDTh?  
Character

#### Overlapped? Default

No  
.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: <b>0.0</b> and <b>13.0</b>

---

#### **Examples**

```
CALC:TRAN:TIME:KBES 10  
calculate2:transform:time:kbessel 13
```

---

#### **Query Syntax Return Type**

CALCulate<cnum>:TRANSform:TIME:KBESsel?  
Character

---

#### **Overlapped? Default**

No  
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? Default**

No  
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: <b>0</b> and $2 * [(number\ of\ points - 1) / frequency\ span]$ <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> 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 Return Type**

CALCulate<cnum>:TRANSform:TIME:SPAN?  
Character

---

#### **Overlapped? Default**

No  
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:  
 $\pm$  (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  
Return Type**

CALCulate<cnum>:TRANSform:TIME:STARt?  
Character

**Overlapped?  
Default**

No  
-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.

<ONIOFF> **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  
Return Type**

CALCulate<cnum>:TRANSform:TIME:STATe?  
Boolean (1 = ON, 0 = OFF)

**Overlapped?  
Default**

No  
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:  
 $\pm$  (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  
Return Type**

CALCulate<cnum>:TRANSform:TIME:STOP?  
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:time 15 ps
```

#### **Query Syntax Return Type**

CALCulate<cnum>:TRANSform:TIME:STEP:RTIME?  
Character

#### **Overlapped? Default**

No  
.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 Return Type**

CALCulate<cnum>:TRANSform:TIME:STIMulus?  
Character

#### **Overlapped? Default**

No  
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  
Return Type**

CALCulate<cnum>:TRANSform:TIME[:TYPE]?  
Character

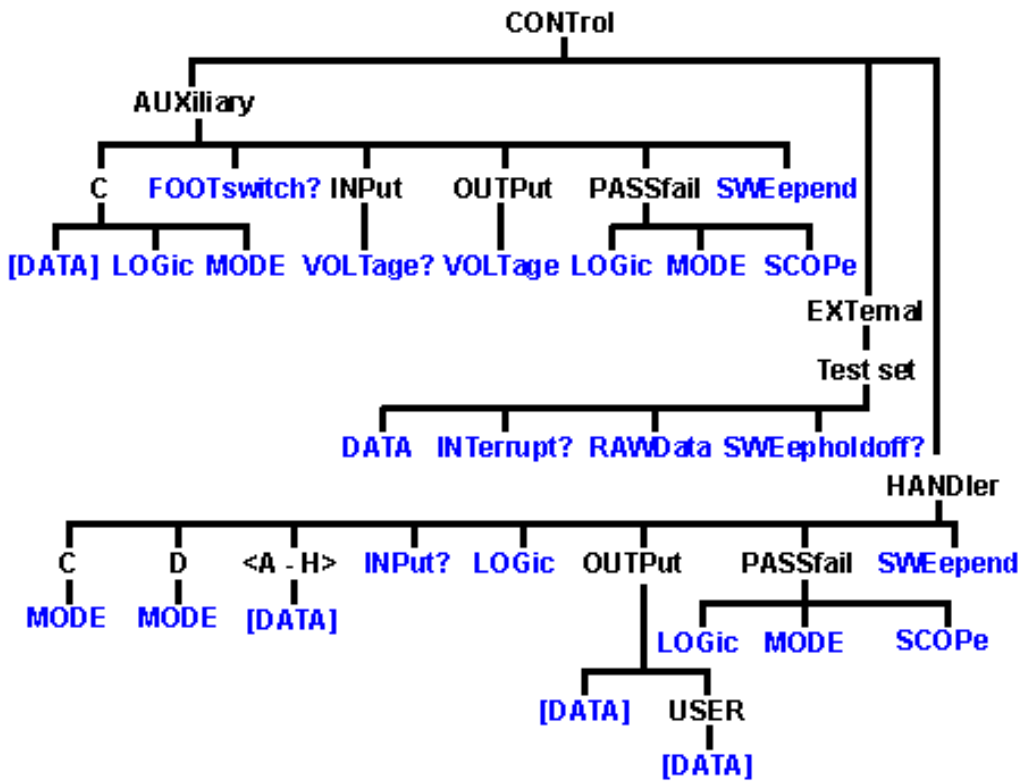
**Overlapped?  
Default**

No  
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 Return Type**

CONTRol:AUXiliary:C:DATA?  
Integer

#### **Overlapped? Default**

No  
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**

```
CONTRol:AUXiliary:C:LOG POS 'Positive logic is applied to Port C data.
```

#### **Query Syntax Return Type**

CONTRol:AUXiliary:C:LOGic?  
Character

#### **Overlapped? Default**

No  
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**  
**Return Type**

CONTROL:AUXiliary:C:MODE?  
Character

**Overlapped?**  
**Default**

No  
INPut

---

**CONTROL:AUXiliary:FOOTswitch?**

(Read-Write) 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?**  
**Default**

No  
False (0) - Released

---

**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**  
**Overlapped?**  
**Default**

REAL  
No  
Not Applicable

---

**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 balue between **-10** and **+10** volts

**Examples**

```
CONT:AUX:OUTP:VOLT 2
control:auxiliary:output:voltage 2
```

**Query Syntax**

CONTROL:AUXiliary:OUTPut<out>:VOLTage?

**Return Type**

'Reads the output DAC voltage  
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.  
**NOWait** - 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  
Return Type**

CONTROL:AUXiliary:SWEepend?  
Character

---

**Overlapped?  
Default**

No  
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 (strokes 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?  
Default**

No  
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:RAWDData <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:RAWDData?`  
**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.  
 Integer

**Return Type**

**Overlapped?**  
**Default**

No  
 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?**  
**Default**

No  
 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**  
**Return Type**

CONTrol:HANDler:C:MODE?  
 Character

**Overlapped?**  
**Default**

No  
 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  
Return Type**

CONTrol:HANDler:D:MODE?  
Character

**Overlapped?  
Default**

No  
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  
Return Type**

CONTrol:HANDler:<port>:DATA?  
Integer

**Overlapped?  
Default**

No  
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 Return Type

CONTrol:HANDler:LOGic?  
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**  
**Return Type**  
CONTrol:HANDler:OUTPut<num>:DATA?  
Integer (0 or 1)

---

**Overlapped?**  
**Default**  
No  
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**  
**Return Type**  
CONTrol:HANDler:OUTPut<num>:USER:DATA?  
Integer (0 or 1)

---

**Overlapped?**  
**Default**  
No  
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**  
**Return Type**  
CONTrol:HANDler:PASSfail:LOGic?  
Character

---

**Overlapped?**  
**Default**  
No  
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.  
**NOWait**- 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  
Return Type**

CONTROL:HANDLER:SWEepend?  
Character

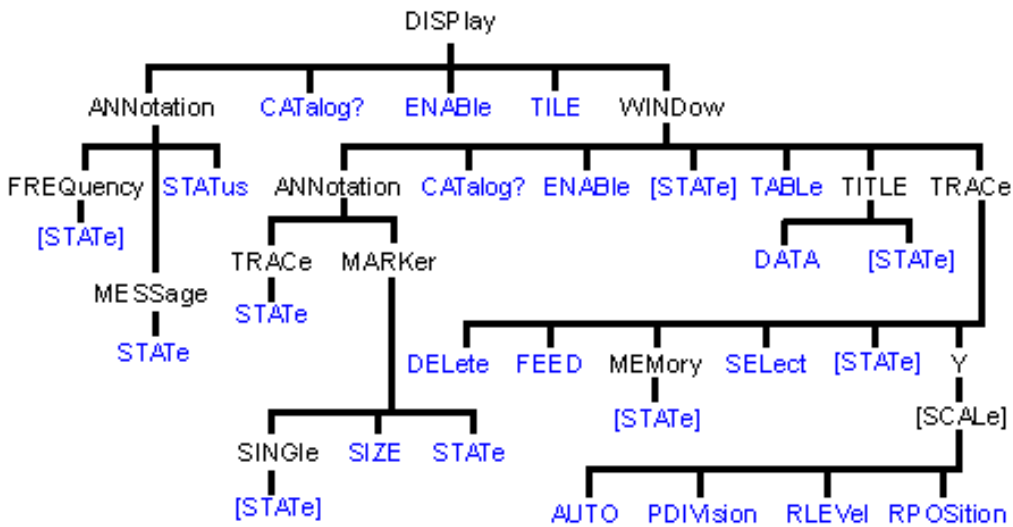
**Overlapped?  
Default**

No  
SWEep



**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

<b>Query Syntax</b>	DISPlay:ANNotation:FREQuency[:STATe]?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	<b>ON (1)</b>

### DISPlay:ANNotation:MESSAge:STATe <ON | OFF>

(Read-Write) Enables and disables error pop-up messages on the display.

#### Parameters

<ON   OFF>	<b>ON</b> (or 1) - enables error pop-up messages <b>OFF</b> (or 0) - disables error pop-up messages
------------	--

#### Examples

```
DISP:ANN:MESS:STAT ON
display:annotation:message:state off
```

<b>Query Syntax</b>	DISPlay:ANNotation:MESSAge:STATe?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)

<b>Overlapped?</b>	No
<b>Default</b>	<b>ON (1)</b>

### 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>	<b>ON</b> (or 1) - turns status bar ON. <b>OFF</b> (or 0) - turns status bar OFF.
------------	--

#### Examples

```
DISP:ANN:STAT ON
display:annotation:status off
```

<b>Query Syntax</b>	DISPlay:ANNotation:STATus?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)

<b>Overlapped?</b>	No
<b>Default</b>	<b>OFF</b>

### DISPlay:CATalog?

(Read-only) Returns the existing Window numbers.

<b>Return Type</b>	String of Character values, separated by commas
<b>Example</b>	<b>Two windows with numbers 1 and 2 returns:</b> "1,2"

<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (or 1) - turns the display ON. <b>OFF</b> (or 0) - turns the display OFF.
------------	--

<b>Examples</b>	DISP:ENAB ON display:enable off
<b>Query Syntax</b>	DISPlay:ENABle?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	ON

### DISPlay[:TILE]

(Write-only) Tiles the windows on the screen.

<b>Examples</b>	DISP display:tile
<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (or 1) - show a single marker per trace <b>OFF</b> (or 0) - show up to 4 markers per active trace

<b>Examples</b>	DISP:WIND:ANN:MARK:SING ON display>window:annotation:marker:single off
<b>Query Syntax</b>	DISPlay:WINDow:ANNOtation:MARKer:SINGle?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>NORMAL</b>   <b>LARGE</b>

<b>Examples</b>	DISP:WIND:ANN:MARK:SIZE LARG display>window:annotation:marker:size normal
<b>Query Syntax</b>	DISPlay:WINDow:ANNOtation:MARKer:SIZE?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	NORMAL

### DISPlay:WINDow<wnum>:ANNOtation:MARKer:STATe <ONIOFF>

(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   OFF>	<b>ON</b> (or 1) - turns marker data ON. <b>OFF</b> (or 0) - turns marker data OFF.

<b>Examples</b>	DISP:WIND:ANN:MARK:STAT ON display>window:annotation:marker:state off
<b>Query Syntax</b>	DISPlay:WINDow:ANNotation:MARKer:STATe?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	ON

**DISPlay:WINDow<wnum>:ANNotation:TRACe:STATe <ONIOFF>**

(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.

<b>Examples</b>	DISP:WIND:ANN:TRAC:STAT ON display>window:annotation:trace:state off
-----------------	---

<b>Query Syntax</b>	DISPlay:WINDow:ANNotation:TRACe:STATe?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)

<b>Overlapped?</b>	No
<b>Default</b>	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

<b>Example</b>	<b>Window 1 with four traces:</b> DISPlay:WINDow1:CATalog? <b>Returns:</b> "1,2,3,4"
----------------	---

<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	DISP:WIND:ENABle ON display>window1:enable off
-----------------	---

<b>Query Syntax</b>	DISPlay:WINDow<wnum>:ENABle?
<b>Return Type</b>	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 Return Type**

DISPlay:WINDow<wnum>[:STATe]?  
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 SEGMENT  
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 Return Type**

DISPlay:WINDow<wnum>:TITLe:DATA?  
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**  
**Return Type**

DISPlay:WINDow<wnum>:TITLe[:STATe]?  
Boolean (1 = ON, 0 = OFF)

**Overlapped?**  
**Default**

No  
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?**  
**Default**

No  
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?**  
**Default**

No  
"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>  
<ON | OFF>

Any existing trace number; if unspecified, value is set to 1  
**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**  
**Return Type**

DISPlay:WIND<wnum>:TRACe<tnum>:MEMory[:STATe]?  
 Boolean (1 = ON, 0 = OFF)

**Overlapped?**  
**Default**

No  
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?**  
**Default**

No  
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**  
**Return Type**

DISPlay:WIND<wnum>:TRACe<tnum>[:STATe]?  
 Boolean (1 = ON, 0 = OFF)

**Overlapped?**  
**Default**

No  
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

<b>Overlapped?</b>	No
<b>Default</b>	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 ( <b>1 to 4</b> ); 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 Return Type

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:PDIVision?  
Character

<b>Overlapped?</b>	No
<b>Default</b>	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 ( <b>1 to 4</b> ); 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 Return Type

DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RLEVel?  
Character

<b>Overlapped?</b>	No
<b>Default</b>	NA

---

### DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RPOStion <num>

(Read-Write) Sets the **Reference Position** of the specified trace in the specified window

#### Parameters

<wnum>	Any existing window number ( <b>1 to 4</b> ); 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

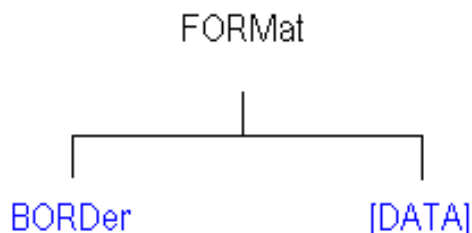
```
DISP:WIND:TRAC:Y:RPOS 0
display:window2:trace2:y:rposition maximum
```

<b>Query Syntax</b>	DISPlay:WINDow<wnum>:TRACe<tnum>:Y[:SCALe]:RPOSition?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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 Return Type

FORMat:DATA?  
Character,Character

---

### Overlapped? Default

No  
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

---

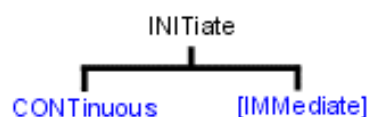
### Overlapped? Default

No  
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 Return Type

INITiate:CONTInuous?  
 Boolean (1 = ON, 0 = OFF)

---

#### Overlapped? Default

No  
 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? Default

Yes  
 Not applicable

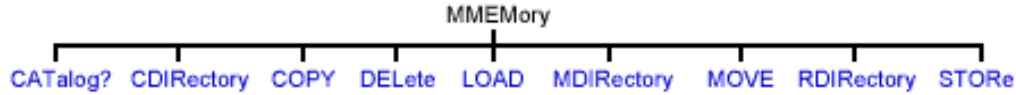
---



## 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

<b>&lt;char&gt;</b>	The type of files to list. Choose from: <b>STATE</b> - Instrument states (.sta) <b>CORRection</b> - Calibration Data (.cal) <b>CSTate</b> - Instrument state and Calibration data (.cst) If unspecified then ALL file types (even unknown types) are listed.
<b>&lt;folder&gt;</b>	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

---

<b>Overlapped?</b>	No
<b>Default</b>	Not applicable

---

### MMEMory:CDIRectory <folder>

(Read-Write) Changes the folder name.

#### Parameters

<b>&lt;folder&gt;</b>	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**  
**Return Type**

MMEMory:CDIRectory? 'Returns the current folder name  
String

---

**Overlapped?**  
**Default**

No  
'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?**  
**Default**

No  
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?**  
**Default**

No  
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'
```

---

<b>Query Syntax</b>	Not applicable
<b>Overlapped?</b>	No
<b>Default</b>	Not applicable

---

### **MMEemory: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

---

### **MMEemory: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

---

### **MMEemory: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

---

### **MMEemory: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.

String - Name of any valid file that is not already in existence.

<file>

**Examples**

```
MMEM:STOR:STAT 'myState'
mmemory:store 'c:\bin\myState.sta'
```

**Query Syntax**

Not applicable

**Overlapped?  
Default**

No  
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  
Return Type**

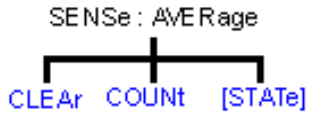
OUTPut[:STaTe]?  
Boolean (1 = ON, 0 = OFF)

**Overlapped?  
Default**

No  
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<num>:AVERage:CLEAr

(Write-only) Clears and restarts averaging of the measurement data. Must also set SENS:AVER[:STATe] ON

#### Parameters

<num> 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<num>:AVERage:COUNT <num>

(Read-Write) Sets the number of measurement sweeps to combine for an average. Must also set SENS:AVER[:STATe] ON

#### Parameters

<num> 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<num>:AVERage:COUNT?

#### Return Type

Character

#### Overlapped?

No

#### Default

1

---

### SENSe<num>:AVERage[:STATe] <ON | OFF>

(Read-Write) Turns trace averaging ON or OFF.

#### Parameters

<num> 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
<b>Query Syntax</b>	SENSe<cnum>:AVERAge[:STATe]?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	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 is entered.)

#### Examples

```
SENS:BWID 1KHZ
sense2:bandwidth:resolution 1000
```

#### Query Syntax Return Type

SENSe<cnum>:BANDwidth | BWIDth[:RESolution]?  
Character

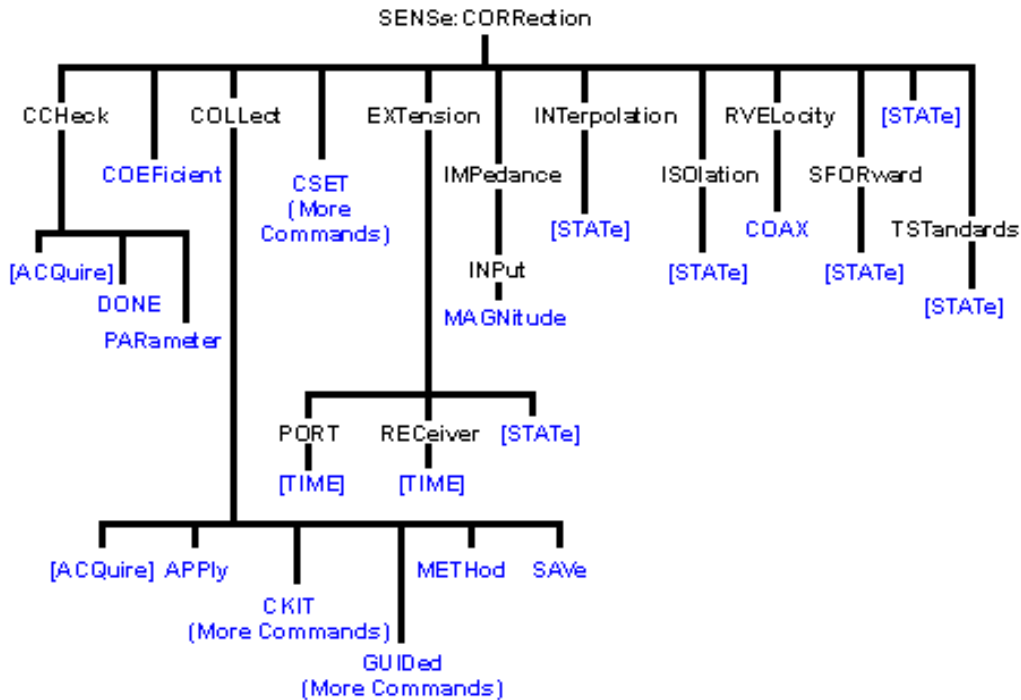
#### Overlapped? Default

No  
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<num>:CORRection:CCheck[:ACQuire] <char>[,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

<num>

Any existing channel number. If unspecified, value is set to 1.

<char>

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. If this argument is not used, the default is **CHAR0**.

#### <char>

**CHAR0** Factory characterization (data that was stored in the ECal module by Agilent)

**CHAR1** User characterization (data that was written to the module by the User Characterization feature on the PNA)

#### 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:COEFFicient <term>[,port]**

(Write-only) Displays the selected error term as trace data. Correction must be turned on.

**Note:** This command displays error terms for only **Channel 1** measurements

#### **Parameters**

<cnum> Any existing channel number. If unspecified, value is set to 1  
<term> Error term to be displayed:

Specify this: to get this **Term...**

<b>1</b>	Directivity
<b>2</b>	Source Match
<b>3</b>	ReflectionTracking
<b>4</b>	Isolation
<b>5</b>	Load Match
<b>6</b>	Transmission Tracking

---

[port] Optional Parameter - Port number of the source drive. If unspecified, value is set to 1.

---

#### **Examples**

```
SENS:CORR:COEF 1  
'Displays the directivity term for port 1 source
```

```
SENS:CORR:COEF 2,2
'Displays the source match for port 2 source
```

<b>Query Syntax</b>	Not applicable
<b>Overlapped?</b>	No
<b>Default</b>	Not applicable

### **SENSe<num>:CORRection:COLLect[:ACQuire] <char>[,char]**

(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 a measurement using CALC:PAR:SEL. You can select one measurement for each channel.

#### **Parameters**

<num>	Any existing channel number. If unspecified, value is set to 1
<char>	Choose from: <b>&lt;char&gt; Measures the standards associated with these class labels:</b> <b>STAN1</b> S11A and S22A <b>STAN2</b> S11B and S22B <b>STAN3</b> S11C and S22C <b>STAN4</b> S21T and S12T - usually the THRU standard <b>STAN5</b> Generic Isolation; not associated with calibration kit definition <b>ECALA</b> ECAL module A <b>ECALB</b> ECAL module B <b>SLSET</b> 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. <b>SLDONE</b> Computes the sliding load using a circle fit algorithm.
[char]	Optional argument. For mechanical calibration kits, this argument is for specifying the standard identified in the SENS:CORR:COLL:CKIT:ORDer list to be acquired. If this argument is not used, the default is <b>SST1</b> . <b>&lt;char&gt; Measures the standards associated with these class labels:</b> <b>SST1</b> First standard in the order list <b>SST2</b> Second standard in the order list <b>SST3</b> Third standard in the order list <b>SST4</b> Fourth standard in the order list <b>SST5</b> Fifth standard in the order list <b>SST6</b> Sixth standard in the order list <b>SST7</b> Seventh standard in the order list  For ECal, this argument is for specifying which characterization within the ECal module will be used for the acquire. If this argument is not used, the default is <b>CHAR0</b> . <b>&lt;char&gt; Measures the standards associated with these class labels:</b> <b>CHAR0</b> Factory characterization (data that was stored in the ECal module by Agilent) <b>CHAR1</b> User characterization (data that was written to the module by the User Characterization feature on the PNA)

#### **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<num>: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<num>:CORRection:COLLect:SAVE calculates and then automatically applies error terms after you use SENS:CORR:COLL:ACQuire to measure cal standards.

#### Parameters

<num> 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<num>: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

<num> 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



<b>REFL3</b>	Full 1 port
<b>TRAN1</b>	Response Thru
<b>TRAN2</b>	Response Thru and Isolation
<b>SPARSOLT</b>	Full SOLT 2 port

<b>Examples</b>	<code>SENS:CORR:COLL:METH REFL1</code> <code>sense2:correction:collect:method sparsolt</code>
<b>Query Syntax</b>	<code>SENSe&lt;num&gt;:CORRection:COLLect:METhod?</code>
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

### **SENSe<num>: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**

<num> Any existing channel number. If unspecified, value is set to 1

<b>Examples</b>	<code>SENS:CORR:COLL:SAVE</code> <code>sense2:correction:collect:save</code>
<b>Query Syntax</b>	Not applicable
<b>Overlapped?</b>	No
<b>Default</b>	Not applicable

### **SENSe<num>: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**

<num> 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**

<b>Examples</b>	<code>SENS:CORR:EXT:PORT 2MS</code> <code>sense2:correction:extension:port2 .00025</code>
<b>Query Syntax</b>	<code>SENSe&lt;num&gt;:CORRection:EXTension:PORT&lt;pnum&gt; [:TIME]?</code>
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	0

---

**SENSe<num>: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**

<num>	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: <b>1</b> for Receiver A <b>2</b> for Receiver B
<num>	The electrical length in seconds; may include suffix. Choose a number between: <b>-10</b> and <b>10</b>

---

**Examples**

```
SENS:CORR:EXT:REC 2MS  
sense2:correction:extension:receiver2:time .00025
```

---

**Query Syntax  
Return Type**

SENSe<num>:CORRection:EXTension:RECeiver<Rnum> [:TIME]?  
Character

---

**Overlapped?  
Default**

No  
0

---

**SENSe<num>: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**

<num>	Any existing channel number. If unspecified, value is set to 1
<ON   OFF>	<b>ON</b> (or 1) - turns port extensions ON. <b>OFF</b> (or 0) - turns port extensions is OFF.

---

**Examples**

```
SENS:CORR:EXT ON  
sense2:correction:extension:state off
```

---

**Query Syntax  
Return Type**

SENSe<num>:CORRection:EXTension[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

**Overlapped?  
Default**

No  
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:

**Return Type** INPut:MAGNitude?  
Character

**Overlapped?** No  
**Default** 50

---

### **SENSe<num>:CORRection:INTerpolation[: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**

<num> 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:interpolation:state off
```

#### **Query Syntax Return Type**

SENSe<num>:CORRection:INTerpolation[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

**Overlapped?** No  
**Default** ON

---

### **SENSe<num>: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**

<num> 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 Return Type**

SENSe<num>:CORRection:ISOLation[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

**Overlapped?** No  
**Default** OFF - (Isolation disabled)

---

### **SENSe<num>:CORRection:RVELocity:COAX <num>**

(Read-Write) Sets the velocity factor to be used with Electrical Delay and Port Extensions.

#### **Parameters**

<num> 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<num>: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<num>: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**

<num> 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<num>: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
```

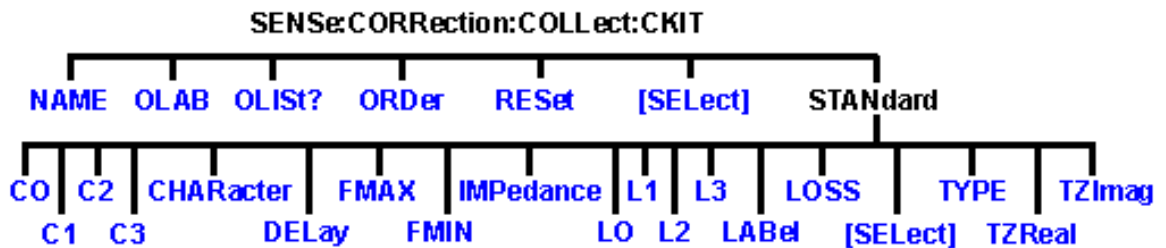
See an example using this command

<b>Query Syntax</b>	SENSe:CORRection:TSTandards[:STATe]?
<b>Overlapped?</b>	No
<b>Default</b>	ON



## 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: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

```
SENSe: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>

<class>	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	S12T	Thru standard
<b>3-port analyzers only</b>		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru standard
13	S23T	Thru standard
14	S31T	Thru standard
15	S13T	Thru standard

<name>

---

### Examples

---

### Return Type

Overlapped?

Default

---

## SENSe:CORRection:COLLect:CKIT:OLIST[class]?

(Read-only) Returns seven values of standards that are assigned to the specified class.

### Parameters

<class>

<class>	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	S12T	Thru standard
<b>3-port analyzers only</b>		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru standard
13	S23T	Thru standard
14	S31T	Thru standard
15	S13T	Thru standard

<class>

---

### Examples

---

### Return Type

Overlapped?  
Default

**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 15**  
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 standard
5	S22A	Reflection standard
6	S22B	Reflection standard
7	S22C	Reflection standard
8	S12T	Thru standard
<b>3-port analyzers only</b>		
9	S33A	Reflection standard
10	S33B	Reflection standard
11	S33C	Reflection standard
12	S32T	Thru standard
13	S23T	Thru standard

	14	S31T	Thru standard
	15	S13T	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?**  
**Default**

No  
 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**  
**Overlapped?**  
**Default**

Not Applicable  
 No  
 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
5	User Defined 5



6	User Defined 6
7	User Defined 7
8	User Defined 8
9	User Defined 9
10	User Defined 10
99	ECAL module

---

**Examples**

```
SENS:CORR:COLL:CKIT 2
sense2:correction:collect:ckit:select 7
```

---

**Query Syntax  
Return Type**

SENSe:CORRection:COLLect:CKIT?  
Character

---

**Overlapped?  
Default**

No  
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  
Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:C0?  
Character

---

**Overlapped?  
Default**

No  
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  
Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:C1?  
Character

---

**Overlapped?  
Default**

No  
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

---

<b>Examples</b>	The following two commands set C2=(-15) picofarads: SENS:CORR:COLL:CKIT:STAN:C2 -15 sense:correction:collect:ckit:standard:c2 -15
<b>Query Syntax</b> <b>Return Type</b>	SENSe:CORRection:COLLect:CKIT:STANdard:C2? Character
<b>Overlapped?</b> <b>Default</b>	No 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

<b>Examples</b>	The following two commands set C3=15 picofarads: SENS:CORR:COLL:CKIT:STAN:C3 15 sense:correction:collect:ckit:standard:c3 15
-----------------	--

**Query Syntax**  
**Return Type** SENSe:CORRection:COLLect:CKIT:STANdard:C3?  
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

<b>Examples</b>	SENS:CORR:COLL:CKIT:STAN:CHAR COAX sense:correction:collect:ckit:standard:character wave
-----------------	---

**Query Syntax**  
**Return Type** SENSe:CORRection:COLLect:CKIT:STANdard:CHARacter?  
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

<b>Examples</b>	SENS:CORR:COLL:CKIT:STAN:DEL 50e-12 sense2:correction:collect:ckit:standard:delay 50ps
-----------------	---

**Query Syntax** SENSe:CORRection:COLLect:CKIT:STANdard:DELay?

<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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

<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

---

### **SENSe:CORRection:COLLect:CKIT:STANdard:FMIN <num>**

(Read-Write) Sets the minimum 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

<b>Overlapped?</b>	No
<b>Default</b>	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

<b>Overlapped?</b>	No
<b>Default</b>	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?  
Default**

No  
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  
Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:L1?  
Character

---

**Overlapped?  
Default**

No  
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  
Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:L2?  
Character

---

**Overlapped?  
Default**

No  
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

---

<b>Query Syntax</b>	SENSe:CORRection:COLLect:CKIT:STANdard:L3?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	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 <b>1</b> and <b>12</b> characters long. Cannot begin with a numeric.
--------	---

#### **Examples**

```
SENS:CORR:COLL:CKIT:STAN:LAB 'OPEN'  
sense:correction:collect:ckit:standard:label 'Short2'
```

<b>Query Syntax</b>	SENSe:CORRection:COLLect:CKIT:STANdard:LABel?
<b>Return Type</b>	String

<b>Overlapped?</b>	No
<b>Default</b>	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
```

<b>Query Syntax</b>	SENSe:CORRection:COLLect:CKIT:STANdard:LOSS?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	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: <b>1</b> and <b>8</b>
-------	---

#### **Examples**

```
SENS:CORR:COLL:CKIT:STAN 3  
sense:correction:collect:ckit:standard:select 8
```

<b>Query Syntax</b>	SENSe:CORRection:COLLect:CKIT:STANdard[:SELEct]?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	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 Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:TYPE?  
Character

---

#### **Overlapped? Default**

No  
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 Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:TZReal?  
Character

---

#### **Overlapped? Default**

No  
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 Return Type**

SENSe:CORRection:COLLect:CKIT:STANdard:TZImag?  
Character

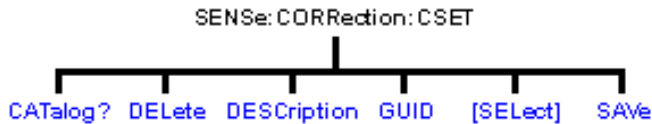
---

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<num>: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

<num> 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<num>: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

<num> 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  
Return Type**

SENSe<cnum>:CORRection:CSET:DESCription?  
String

---

**Overlapped?  
Default**

No  
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?  
Default**

No  
Not Applicable

---

**SENSe<cnum>:CORRection:CSET[:SElect] <char>**

(Read-Write) Restores a correction data set from memory. The file name is "CSETx.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 "at00x.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  
Return Type**

SENSe<cnum>:CORRection:CSET[:SELEct]?  
Character

---

**Overlapped?  
Default**

No  
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 "**CSETx.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 "**at00x.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

---

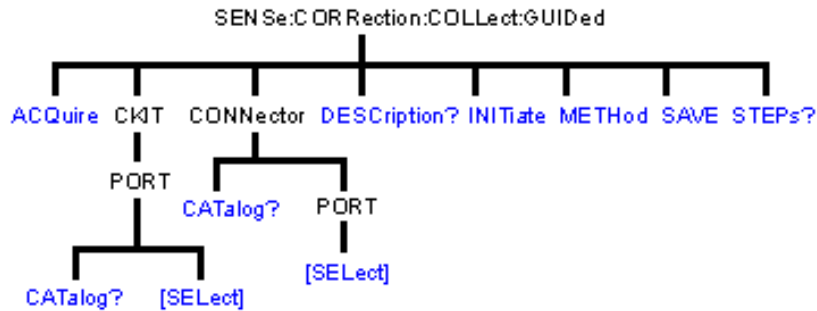
**Overlapped?  
Default**

No  
Not applicable

---

**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 Return Type

Not Applicable  
 Character

#### Overlapped? Default

No  
 Not Applicable

### SENSe<cnum>:CORRection:COLLect:GUIDed:CKIT:PORT<pnum>:CATalog?

(Read-only) Returns a comma-separated list of valid kits for each port. 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

<b>Overlapped?</b>	No
<b>Default</b>	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 '85055A'
```

**Query Syntax  
Return Type**

SENSe:CORRection:COLLect:GUIDed:CKIT:PORT<pnum>[:SElect]?  
String - If the <kit> parameter was incorrectly entered while writing, an error will be returned.

<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

### **SENSe<cnum>:CORRection:COLLect:GUIDed:CONNector:CATalog?**

(Read-only) Returns a comma-separated list of valid connectors. Use an item from the returned list to specify a connector for the SENS:CORR:COLL:GUID:CONN:PORT:SEL command.

**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
```

**Return Type** String

<b>Overlapped?</b>	No
<b>Default</b>	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**

<cnnum>	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<cnnum>:CORRection:COLLect:GUIDed:CONNector:PORT<pnum>[:SElect]?
```

**Return Type**

String

**Overlapped?**

No

**Default**

Not Applicable

**SENSe<cnnum>:CORRection:COLLect:GUIDed:DESCription? <step>**

(Read-only) Returns the connection description for the specified calibration step.

**Parameters**

<cnnum>	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<cnnum>:CORRection:COLLect:GUIDed:INITiate**

(Write-only) Initiates a guided calibration by creating a new Cal Set.

The analyzer 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

**Examples**

```
SENS:CORR:COLL:GUID:INIT
sense2:correction:collect:guided:initiate
```

**Query Syntax**

Not Applicable

**Overlapped?**

No

**Default**

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** - Informs 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).

**ADAPRemove** - 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.

**UNKNown** - 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.

**Examples**

```
SENS:CORR:COLL:GUID:METh
sense2:correction:collect:guided:method unkn
```

**Query Syntax**

Not Applicable

**Overlapped?**

No

**Default**

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
```

<b>Query Syntax</b>	Not Applicable
<b>Overlapped?</b>	No
<b>Default</b>	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: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

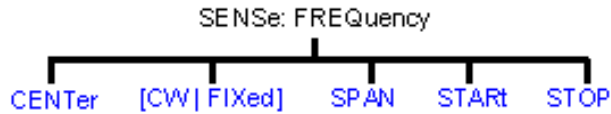
#### Default

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 Return Type

```

SENSe<cnum>:FREQuency:CENTer?
Character
  
```

#### Overlapped? Default

No  
 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 Return Type

```

SENSe<cnum>:FREQuency[:CW | :FIXed]?
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 Return Type

SENSe<cnum>:FREQUENCY:SPAN?  
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 Return Type

SENSe<cnum>:FREQUENCY:START?  
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

---

<b>Query Syntax</b>	SENSe<cnum>:FREQuency:STOP?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>ARECeiver</b> - receiver A <b>BRECeiver</b> - receiver B
<num>	Choose from: <b>0</b> to <b>35</b> 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:  
attenuation breceiver,30
```

---

<b>Query Syntax</b>	SENSe<cnum>:POWer :ATTenuation? <rec>
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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

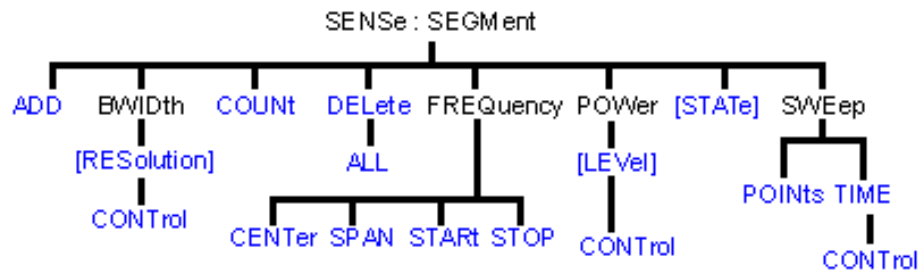
### Default

Not applicable



## 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<num>:SEGMENT<snum>:ADD

(Write-only) Adds a segment.

### Parameters

<num>

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
```

<b>Query Syntax</b>	Not applicable. Use Sense:Segment:Count to determine the number of segments in a trace.
<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

### **SENSE<cnum>:SEGMENT<snum>:BWIDTh[:RESolution] <num>**

(Read-Write) Sets the IF Bandwidth 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: <b>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  </b> 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 is entered.) <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> instead of a numeric parameter. See SCPI Syntax for more information.

#### **Examples**

```
SENS:SEGM:BWID 1KHZ  
sense2:segment2:bwid:resolution max
```

<b>Query Syntax</b>	SENSe<cnum>:SEGMENT<snum>:BWIDTh[:RESolution]?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>ON</b> (or 1) - turns Bandwidth control ON. Bandwidth can be set for each segment <b>OFF</b> (or 0) - turns Bandwidth control OFF. Use channel bandwidth setting

#### **Examples**

```
SENS:SEGM:BWID:CONTRol ON  
sense2:segment:bwid:control off
```

<b>Query Syntax</b>	SENSe<cnum>:SEGMENT:BWIDTh[:RESolution]:CONTRol?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)

<b>Overlapped?</b>	No
<b>Default</b>	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
--------	--

<b>Examples</b>	SENS:SEGM:COUNT? sense2:segment:count?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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

<b>Examples</b>	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

<b>Examples</b>	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.

<b>Examples</b>	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 Return Type**

SENSe<cnum>:SEGMent<snum>:FREQUency:SPAN?  
Character

#### **Overlapped? Default**

No  
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 Return Type**

SENSe<cnum>:SEGMent<snum>:FREQUency:START?  
Character

#### **Overlapped? Default**

No  
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<num>:SEGMENT<num>[:STATe] <ON | OFF>**

(Read-Write) Turns the specified sweep segment ON or OFF.

**Parameters**

<num> Any existing channel number. If unspecified, value is set to 1  
 <num> 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  
Return Type**

SENSe<num>:SEGMENT[:STATe]? <num>  
 Boolean (1 = ON, 0 = OFF)

**Overlapped?  
Default**

No  
 OFF

**SENSe<num>:SEGMENT<num>:SWEep:POINTs <num>**

(Read-Write) Sets the number of data points for the specified sweep segment.

**Parameters**

<num> Any existing channel number. If unspecified, value is set to 1  
 <num> 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 **1601**. 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  
Return Type**

SENSe<num>:SEGMENT<num>:SWEep:POINTs?  
 Character

**Overlapped?  
Default**

No  
 201

**SENSe<num>:SEGMENT<num>:SWEep:TIME <num>**

(Read-Write) Sets the time the analyzer takes to sweep the specified sweep segment.

**Parameters**

<num> Any existing channel number. If unspecified, value is set to 1  
 <num> 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  
Return Type**

SENSe<num>:SEGMENT<num>:SWEep:TIME?  
 Character

**Overlapped?  
Default**

No  
 Not Applicable

**SENSe<num>:SEGMENT:SWEep:TIME:CONTROL <ON | OFF>**

(Read-Write) Specifies whether Sweep Time can be set independently for each sweep segment.

**Parameters**

<num> 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<num>:SEGMENT:SWEep:TIME:CONTRol?  
 Boolean (1 = ON, 0 = OFF)

**Overlapped?**

No

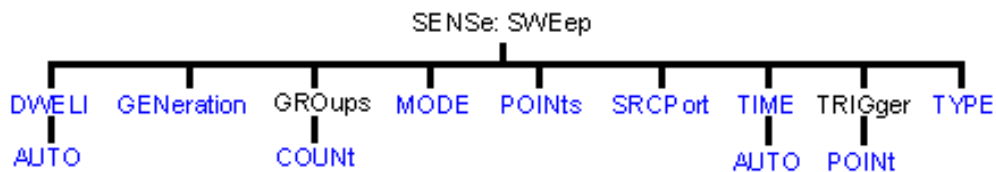
**Default**

OFF



**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<num>:SWEep:DWELI <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**

<num> 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:dwel min

**Query Syntax**  
**Return Type**

SENSe<cnum>:SWEep:DWELI?  
Character

**Overlapped?**  
**Default**

No  
0 - (**Note:** dwell time set to 0 is the same as dwell:auto ON)

**SENSe<cnum>:SWEep:DWELI: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:dwel:auto off

**Query Syntax**  
**Return Type**

SENSe<cnum>:SWEep:DWELI:AUTO?  
Boolean (1 = ON, 0 = OFF)

**Overlapped?**  
**Default**

No  
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 each 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**  
**Return Type**

SENSe<cnum>:SWEep:GENeration?  
Character

**Overlapped?**  
**Default**

No  
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:COUNT 10

```
sense2:sweep:groups:count 50
```

---

<b>Query Syntax</b>	SENSe<cnum>:SWEep:GROups:COUNT?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>HOLD</b> - channel will not trigger <b>CONTInuous</b> - channel triggers indefinitely <b>GROups</b> - channel accepts the number of triggers specified with the last SENS:SWE:GRO:COUN <num>

---

#### Examples

```
SENS:SWE:MODE CONT  
sense2:sweep:mode hold
```

---

<b>Query Syntax</b>	SENSe<cnum>:SWEep:MODE?
<b>Return Type</b>	Character
<b>Overlapped?</b>	<b>YES</b> - SENS:SWE:MODE GROUPS (when INIT:CONT is ON) <b>NO</b> - HOLD and CONTInuous
<b>Default</b>	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 <b>1</b> and <b>1601</b> <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> instead of a numeric parameter. See SCPI Syntax for more information.

---

#### Examples

```
SENS:SWE:POIN 51  
sense2:sweep:points max
```

---

<b>Query Syntax</b>	SENSe<cnum>:SWEep:POINTs?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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>	<b>1</b> - Source power comes out Port 1 <b>2</b> - 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 Return Type

SENSe<cnum>:SWEep:TYPE?  
Character

---

### Overlapped?

No

### Default

LINear

---

List of all commands in this block:  
(Parameters in ***bold italics***)

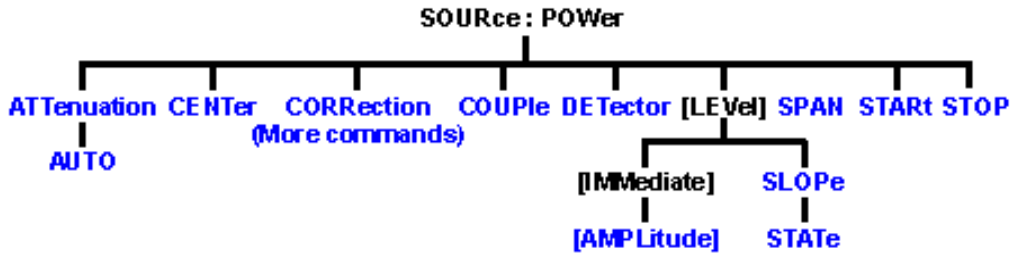
```
:SENSe1:SWEep:DWELL 2ms
:SENSe1:SWEep:DWELL?
:SENSe1:SWEep:DWELL:AUTO ON
:SENSe1:SWEep:DWELL:AUTO?
:SENSe1:SWEep:GENeration STEP
:SENSe1:SWEep:GENeration?
:SENSe1:SWEep:GROups:COUNT 50
:SENSe1:SWEep:GROups:COUNT?
:SENSe1:SWEep:MODE CONT
:SENSe1:SWEep:MODE?
:SENSe1:SWEep:POINTs 201
:SENSe1:SWEep:POINTs?
:SENSe1:SWEep:SRCPort 1
:SENSe1:SWEep:SRCPort?
:SENSe1:SWEep:TIME 2ms
:SENSe1:SWEep:TIME?
:SENSe1:SWEep:TIME:AUTO ON
:SENSe1:SWEep:TIME:AUTO?
:SENSe1:SWEep:TRIGger:POINT ON
:SENSe1:SWEep:TRIGger:POINT?
:SENSe1:SWEep:TYPE LINear
:SENSe1:SWEep:TYPE?
```



## 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 <b>1</b> or <b>2</b> ; If unspecified, value is set to 1.
<num>	Choose a number between <b>0</b> and <b>70</b> 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. <b>Note:</b> This command will accept <b>MIN</b> or <b>MAX</b> instead of a numeric parameter. See SCPI Syntax for more information.

#### Examples

```
SOUR:POW:ATT 10
source2:power:attenuation maximum
```

#### Query Syntax Return Type

```
SOURce<cnum>:POWER<port>:ATTenuation?
Character
```

#### Overlapped? Default

```
No
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 <b>1</b> or <b>2</b> ; If unspecified, value is set to 1.
<ON   OFF>	<b>ON</b> (or 1) - turns coupling ON. The analyzer automatically selects the appropriate attenuation level to meet the specified power level. <b>OFF</b> (or 0) - turns coupling OFF. Attenuation level must be set using SOURce:POWER:ATTenuation <num>.

#### Examples

```
SOUR:POW2:ATT:Auto On
source2:power:
```

attenuation: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<num>:POWER<port>[:LEVel][:IMMediate]  
[:AMPLitude] <num>**

(Read-Write) Sets the RF power output level.

**Parameters**

<num> 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  
Return Type**

SOURce<num>:POWER[:LEVel][:IMMediate][:AMPLitude]?  
Character

---

Overlapped? No  
Default 0 dBm

---

**SOURce<num>:POWER[:LEVel]:SLOPe <int>**

(Read-Write) Sets the RF power slope value.

**Parameters**

<num> 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  
Return Type**

SOURce<num>:POWER[:LEVel]:SLOPe?  
Character

---

Overlapped? No  
Default 0

---

**SOURce<num>:POWER[:LEVel]:SLOPe:STATe <ONIOFF>**

(Read-Write) Turns Power Slope ON or OFF.

**Parameters**

<num> Any existing channel number. If unspecified, value is set to 1  
<ONIOFF> **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  
Return Type**

SOURce<num>:POWER[:LEVel]:SLOPe:STATe?  
Boolean (1 = ON, 0 = OFF)

---

Overlapped? No  
Default OFF

---

---

**SOURce<num>:POWER:SPAN <num>**

(Read-Write) Sets the power sweep span power. Must also set:

SENS:SWE:TYPE POWER and SOURce:POWER:CENTer <num>.

**Parameters**

<num> 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  
Return Type**

SOURce<num>:POWER:SPAN?  
Character

---

**Overlapped?  
Default**

No  
0 dBm

---

**SOURce<num>:POWER:STARt <num>**

(Read-Write) Sets the power sweep start power. Must also set

SENS:SWE:TYPE POWER and SOURce:POWER:STOP <num>.

**Parameters**

<num> 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  
Return Type**

SOURce<num>:POWER:STARt?  
Character

---

**Overlapped?  
Default**

No  
0 dBm

---

**SOURce<num>:POWER:STOP <num>**

(Read-Write) Sets the power sweep stop power. Must also set:

SENS:SWE:TYPE POWER and SOURce:POWER:STARt <num>.

**Parameters**

<num> 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  
Return Type**

SOURce<num>:POWER:STOP?  
Character

---

**Overlapped?  
Default**

No  
0 dBm

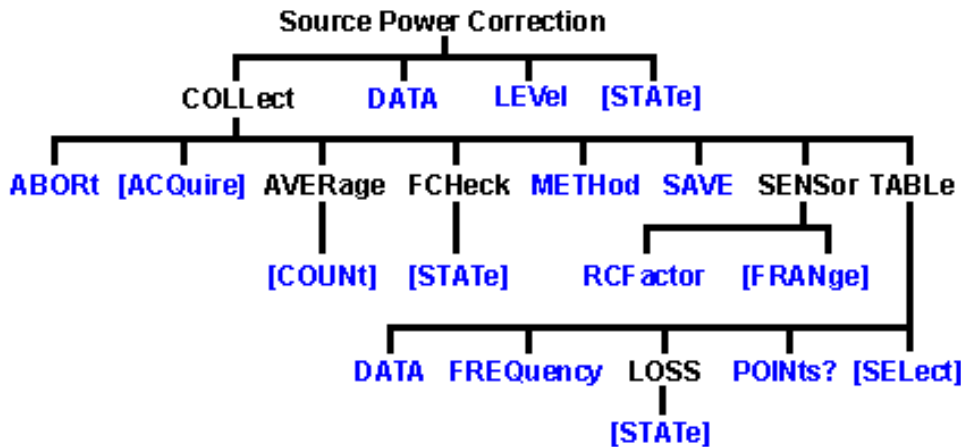
---





## 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<num>:POWER<port>:CORRection:COLlect:ABORT

(Write-only) Aborts a source power calibration sweep that is in progress.

#### Parameters

<num>	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<num>: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**

<num> 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<num>: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**

<num> 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**  
**Return Type**

SOURce:POWER:CORRection:COLLect:AVERAge[:COUNT]?  
Character

---

**Overlapped?**

No

**Default**

1

---

**SOURce<num>:POWER:CORRection:COLLect:FCHeck[:STATe] <ON | OFF>**

(Read-Write) Enables and disables frequency checking of source power cal acquisition sweeps.

**Parameters**

<num> 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.

<b>Examples</b>	SOUR:POW:CORR:COLL:FCH ON source1:power2:correction:collect:fcheck:state off
<b>Query Syntax</b>	SOURce:POWer:CORRection:COLLect:FCHeck[:STATe]?
<b>Return Type</b>	Boolean (1 = ON, 0 = OFF)
<b>Overlapped?</b>	No
<b>Default</b>	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

<b>Examples</b>	SOUR:POW:CORR:COLL:METH PMET source1:power2:correction:collect:method pmeter
<b>Query Syntax</b>	SOURce:POWer:CORRection:COLLect:METhod?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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.

<b>Examples</b>	SOUR:POW:CORR:COLL:SAVE source1:power2:correction:collect:save
-----------------	---

<b>Query Syntax</b>	Not Applicable
<b>Overlapped?</b>	No
<b>Default</b>	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: <b>A</b> - Channel A <b>B</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: <b>A</b> - Channel A <b>B</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**

<num> 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<num>: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<num>: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**

<num> 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<num>: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<num>: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**

<num> 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<num>:POWer:CORRection:COLLect:TABLE:POINts?**

(Read-only) Returns the number of segments that are currently in the selected table.

**Parameters**

<num> Any existing channel number. If unspecified, value is set to 1

<b>Examples</b>	SOUR:POW:CORR:COLL:TABL:POIN? source1:power2:correction:collect:table:points?
-----------------	--

<b>Return Type</b>	Character
--------------------	-----------

<b>Overlapped?</b>	No
<b>Default</b>	0

---

### **SOURce<num>: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**

<num>	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

---

<b>Examples</b>	SOUR:POW:CORR:COLL:TABL ASEN source1:power2:correction:collect:table:select bsensor
-----------------	--

<b>Query Syntax</b>	SOURce:POWER:CORRection:COLLect:TABLE[:SElect]?
<b>Return Type</b>	Character

<b>Overlapped?</b>	No
<b>Default</b>	NONE

---

### **SOURce<num>: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**

<num>	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

---

<b>Examples</b>	SOURce1:POWER2:CORRection:DATA 0.12, -0.34, 0.56
-----------------	--

<b>Query Syntax</b>	SOURce<num>:POWER<port>:CORRection:DATA?
<b>Return Type</b>	Character - One number per trace point

<b>Overlapped?</b>	No
<b>Default</b>	Not Applicable

---

### **SOURce<num>:POWER<port>:CORRection:LEVel <num>**

(Read-Write) Specifies the power level that is expected at the desired reference plane (DUT input or output).

### Parameters

<num>	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 Return Type

SOURce:POWer:CORRection:LEVel?  
Character

---

### Overlapped? Default

No  
0 dBm

---

## SOURce<num>:POWer<port>:CORRection[:STATe] <ONIOFF>

(Read-Write) Enables and disables source power correction for the specified port on the specified channel.

### Parameters

<num>	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.
<ONIOFF>	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 Return Type

SOURce:POWer:CORRection[:STATe]?  
Boolean (1 = ON, 0 = OFF)

---

### Overlapped? Default

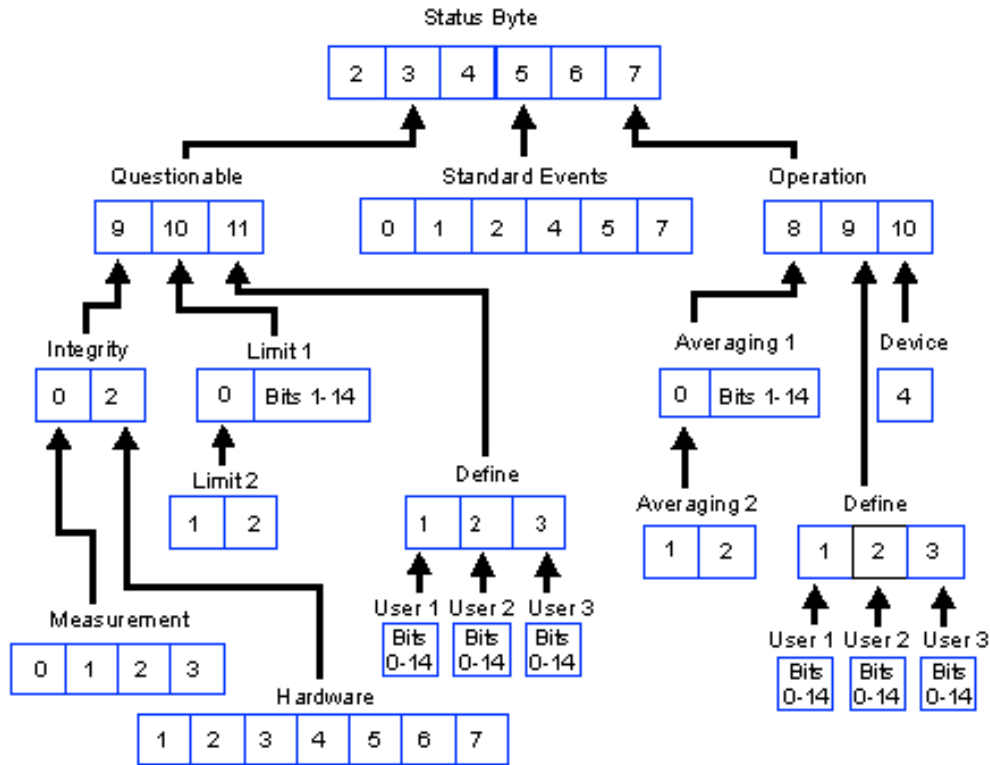
No  
OFF (0)



---

## Status Register Commands

The status registers enable you to query the state of selected events that occur in the analyzer.



- 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 <b>Enable</b> Register.
*SRE <num>	Sets bits in the Service Request <b>Enable</b> 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 <b>questionable</b> event status register is set to 1
4	16	Message Available	the output queue is not empty
5	32	Standard Event Register	any enabled bit in the <b>standard</b> event status register is set to 1



6	64	Summary 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 <b>operation</b> 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	STAT:QUES:NTR 1024
<bits>	
:PTRansition	STAT:QUES:PTR 0
<bits>	

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
9	512	Integrity Reg summary	any enabled bit in the <b>Integrity</b> event register is set to 1
10	1024	Limit Registers summary	any enabled bit in the <b>Limit</b> event registers is set to 1
11	2048	Define Registers summary	any enabled bit in the <b>Define</b> 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 <b>Measurement Integrity</b> event register is set to 1
2	2	Hardware Summary	any bit in the <b>Hardware</b> 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<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.

<b>&lt;keyword&gt;</b>	<b>Example</b>
:CONDition?	STAT:QUES:INT:MEAS:COND?
:ENABLE <bits>	STAT:QUES:INT:MEAS:ENAB 1024
[:EVENT]?	STAT:QUES:INT:MEAS?
:NTRansition <bits>	STAT:QUES:INT:MEAS:NTR 1024
:PTRansition <bits>	STAT:QUES:INT:MEAS:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Channel1	a channel1 setting change has occurred and the data does not yet reflect that change.
1	2	Channel2	a channel2 setting change has occurred and the data does not yet reflect that change.
2	4	Channel3	a channel3 setting change has occurred and the data does not yet reflect that change.
3	8	Channel4	a channel4 setting change has occurred and the data does not yet reflect that change.

**STATus:QUEStionable:LIMit1<keyword>**

Monitors the status of limit line failures and summarizes the Limit2 event register. When a trace fails, or any bit in the Limit2 event register fails, the representative bit is set to 1. These enable bits are set to 1 by default.

**Note:** The '1' on 'LIMit1' is a parameter. If unspecified, the value is set to 1.

<b>&lt;keyword&gt;</b>	<b>Example</b>
:CONDition?	STAT:QUES:LIM1:COND?
:ENABLE <bits>	STAT:QUES:LIM1:ENAB 1024
[:EVENT]?	STAT:QUES:LIM1?
:NTRansition <bits>	STAT:QUES:LIM1:NTR 1024
:NTRansition?	STAT:QUES:LIM1:NTR?
:PTRansition <bits>	STAT:QUES:LIM1:PTR 0
:PTRansition?	STAT:QUES:LIM1:PTR?

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	Limit 2 Reg summary	any point from limit2 event register fails
1	2	Trace 1	any point on trace fails the limit test
2	4	Trace 2	any point on trace fails the limit test

3	8	Trace 3	any point on trace fails the limit test
4	16	Trace 4	any point on trace fails the limit test
5	32	Trace 5	any point on trace fails the limit test
6	64	Trace 6	any point on trace fails the limit test
7	128	Trace 7	any point on trace fails the limit test
8	256	Trace 8	any point on trace fails the limit test
9	512	Trace 9	any point on trace fails the limit test
10	1024	Trace 10	any point on trace fails the limit test
11	2048	Trace 11	any point on trace fails the limit test
12	4096	Trace 12	any point on trace fails the limit test
13	8192	Trace 13	any point on trace fails the limit test
14	16384	Trace 14	any point on trace fails the limit test

**STATus:QUEStionable:LIMit2<keyword>**

Monitors the status of limit line failures. When trace 15 or 16 fails, the representative bit is set to 1. These enable bits are set to 1 by default.

**Note:** The '2' on 'LIMit2' is a parameter. If unspecified, the value is set to 1.

<keyword>	Example
:CONDition?	STAT:QUES:LIM2:COND?
:ENABle <bits>	STAT:QUES:LIM2:ENAB 1024
[:EVENT]?	STAT:QUES:LIM2?
:NTRansition <bits>	STAT:QUES:LIM2:NTR 1024
:PTRansition <bits>	STAT:QUES:LIM2:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
1	2	Trace15	any point on trace fails the limit test
2	4	Trace16	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 <b>USER1</b> event register is set to 1
2	4	USER2	any bit in the <b>USER2</b> event register is set to 1
3	8	USER3	any bit in the <b>USER3</b> 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 <b>ENABLE</b> register.
*ESE <bits>	Sets bits in the standard event <b>ENABLE</b> 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 <b>EVENT</b> 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 <b>in order</b> :  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 <b>OR</b> - 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 <b>OR</b> - 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:  <ul style="list-style-type: none"> <li>• did not follow proper syntax</li> <li>• was misspelled</li> </ul>
7	128	Power ON	was an optional command it does not implement 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 <b>Averaging summary</b> event register is set to 1
9	512	User Defined summary	
10	1024	Device summary	either enabled bit in the <b>Device summary</b> event register is set to 1

---

**STATus:OPERation:AVERaging1<keyword>**

Monitors Averaging of Traces 1 to 14 and summarizes Traces 15 and 16.

Note:The '1' on 'AVERaging1' is a parameter. If unspecified, the value is set to 1.

<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

---

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
0	1	AVER2 summary	any enabled bit in AVERaging2 event register is set to 1
1	2	Trace 1	Averaging is complete
2	4	Trace 2	Averaging is complete
3	8	Trace 3	Averaging is complete
4	16	Trace 4	Averaging is complete
5	32	Trace 5	Averaging is complete
6	64	Trace 6	Averaging is complete
7	128	Trace 7	Averaging is complete
8	256	Trace 8	Averaging is complete
9	512	Trace 9	Averaging is complete
10	1024	Trace 10	Averaging is complete
11	2048	Trace 11	Averaging is complete
12	4096	Trace 12	Averaging is complete
13	8192	Trace 13	Averaging is complete
14	16384	Trace 14	Averaging is complete

---

**STATus:OPERation:AVERaging2<keyword>**

Monitors Averaging of Traces 15 and 16.

**Note:**The '2' on 'AVERaging2' is a parameter. If unspecified, the value is set to 1.

<keyword>	Example
:CONDition?	STAT:OPER:AVER2:COND?
:ENABle <bits>	STAT:OPER:AVER2:ENAB 1024
[:EVENT]?	STAT:OPER:AVER2?
:NTRansition <bits>	STAT:OPER:AVER2:NTR 1024

:PTRansition <bits>

STAT:OPER:AVER2:PTR 0

Bit	Weight	Description	Bit is set to 1 when the following conditions exist:
1	2	Trace 15	Averaging is complete
2	4	Trace 16	Averaging 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 <b>USER1</b> event register is set to 1
2	4	USER2	any bit in the <b>USER2</b> event register is set to 1
3	8	USER3	any bit in the <b>USER3</b> 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 be 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 occurred. These bits remain set until they are read or otherwise cleared.

---

#### **:MAP <bit>,<error>**

Associates a bit in 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.

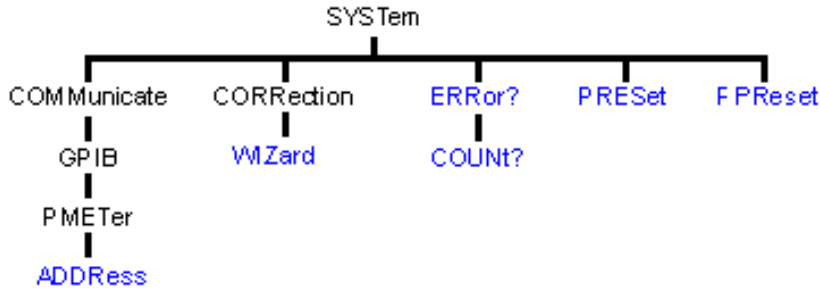
---



## **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:gpiib:pmet: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?**      No  
**Default**            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: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

---

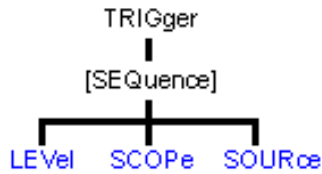


## Trigger Commands

---

Starts or ends a measurement sequence. These commands are an important part of synchronizing measurements.

Change slope to level



- Click on a blue keyword to view the command details.
- See a List of all commands in this block.
- Learn about Triggering

### 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 Return Type

TRIGger[:SEquence]:LEVel?  
Character

#### Overlapped? Default

No  
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. You can use **CURRENT** only when TRIG:SOURCE = **MANUAL**.

#### Examples

```
TRIG:SCOP ALL
trigger:sequence:scope current
```

#### Query Syntax Return Type

TRIGger[:SEquence]:SCOPE?  
Character

#### Overlapped? Default

No  
ALL

### TRIGger[:SEquence]:SOURce <char>

(Read-Write) Sets the source of the sweep trigger signal. This command is a superset 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.

---

<b>Examples</b>	TRIG:SOUR EXT trigger:sequence:source immediate
<b>Query Syntax</b>	TRIGger[:SEQuence]:SOURce?
<b>Return Type</b>	Character
<b>Overlapped?</b>	No
<b>Default</b>	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 ref13"  
    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 ERRMSGSSIZE 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* ErrorMessage);

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("CALCulatel: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:TRACel:FEED 'My_S11'");

// Setup the channel for single sweep trigger
GPIBWrite("INITiatel:CONTinuous OFF;*OPC?");
opc = GPIBRead();
GPIBWrite("SENSEl:SWEep:TRIGger:POINT OFF");

// Set channel parameters
GPIBWrite("SENSEl:SWEep:POINTs 11");
GPIBWrite("SENSEl:FREQuency:START 1000000000");
GPIBWrite("SENSEl:FREQuency:STOP 2000000000");

// Send a trigger to initiate a single sweep
GPIBWrite("INITiatel;*OPC?");
opc = GPIBRead();

// Must select the measurement before we can read the data
GPIBWrite("CALCulatel: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 ErrorMessage[ERRMSGSIZE + 1];
    length = strlen(SCPIcmd) ;

    ibwrt (pna, SCPIcmd, length);
    if (ibsta & ERR)
    {
        strcpy(ErrorMessage, "Unable to write this command to PNA:\n");
        strcat(ErrorMessage, SCPIcmd);

        GPIBCleanup(pna, ErrorMessage);
        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* ErrorMessage)
{
    printf("Error : %s\nibsta = 0x%x iberr = %d (%s)\n",
        ErrorMessage, 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
steplofN = "Step " + va + " of " + dat
GPIB.Write "sens:corr:coll:guid:desc? " + va
prompt = GPIB.Read(80)
Value = MsgBox(prompt, vbOKOnly, steplofN)
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 <ASENsensor | BSENsensor>. 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$(10)" appends an ASCII linefeed character to the
' output, for terminating the write transaction.
myGPIBWrite = viVPrintf(viHandle, strOut + Chr$(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

```

## 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 = viPrintf(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

---

[See Other SCPI Example Programs](#)

---

```

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:EVENt?"
 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 SIDL
- 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
- 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.

---

### 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	<b>no parallel poll capability</b>

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

---

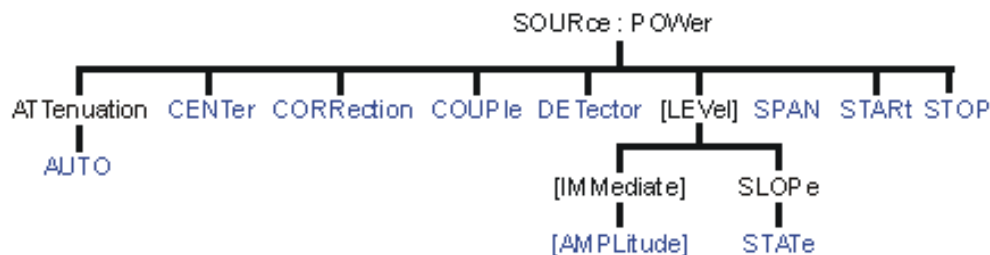
Other Topics about GPIB Concepts

---

### 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 `:CALCULATE`, 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.

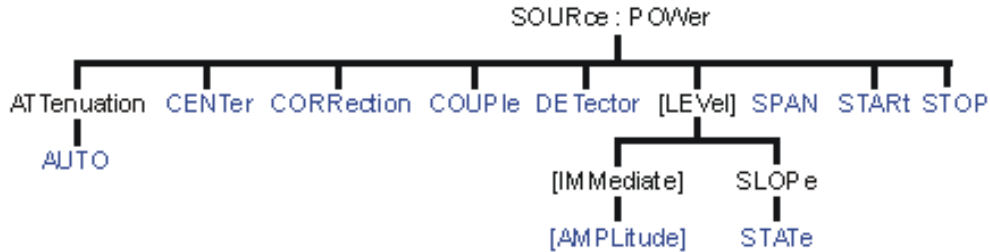
## Command and Query

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
<hr/> <b>Event commands</b> - cause an action to occur inside the analyzer. <hr/>	:INITIATE:IMMEDIATE
<b>Query commands</b> - query only; there is no associated analyzer state to set. <hr/>	:SYSTem:ERRor?
<b>Command and query</b> - 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:BOReDer 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 its 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

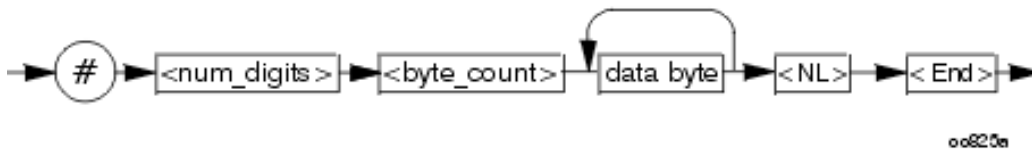
```
GPIB.Write "DISP:WINDow:TITLe:DATA?"
```

```
"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

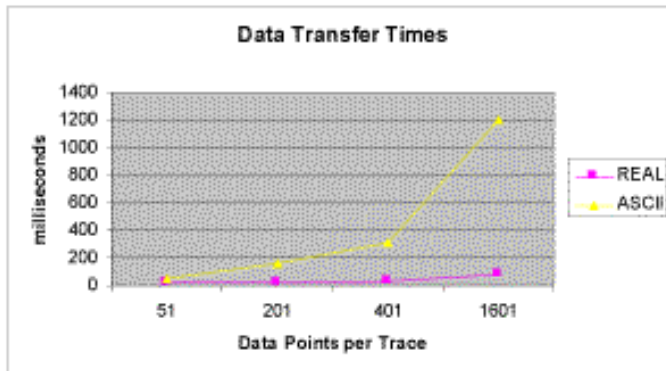
**Transferring Measurement Data**

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.

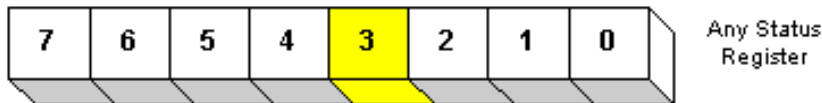
- Polled Bit Method
- Service Request Method
- Setting and Reading Bits in Status Registers
- Positive and Negative Transitions
- Status Commands

---

### 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 interrupts 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 interrupted 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 interrupt 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 interrupt. Use the \*SPE command to determine the instrument that caused the interrupt and then poll the summary registers, and then condition registers to determine the cause of the interrupt.

## 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

<b>Bit</b>	0	1	2	3	4	5	6	7
<b>Weight</b>	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:  
**INITiate: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

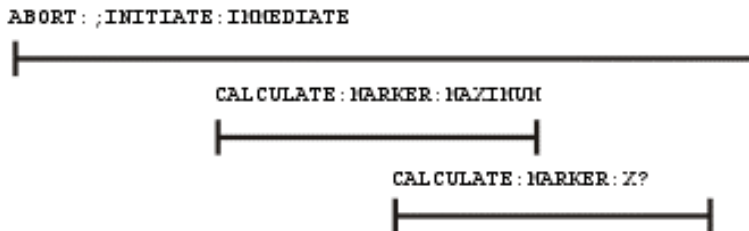
The \*WAI command:

- **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:

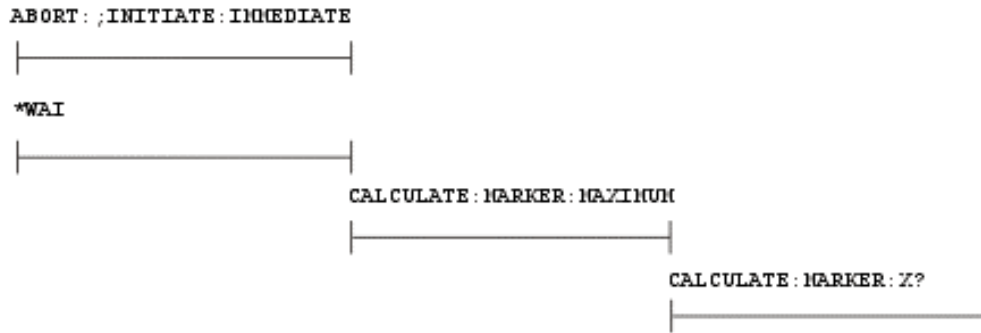


INITIATE:IMMEDIATE is an overlapped command; it allows the immediate processing of the sequential command, CALCULATE:MARKER:SEARCH:MAXIMUM. However, the INITIATE:IMMEDIATE is not considered complete until the measurement is complete. Therefore, the marker searches for maximum amplitude before the measurement 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:SEARCH: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 on the 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 PNA also has the Agilent I/O Libraries installed on it. 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

The following only needs to be done once:

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.

The following must be done every time the PNA is rebooted

1. From the PNA **System** menu, point to **Configure** then click **SICL/GPIB**.
2. In the SCPI/GPIB dialog box, check **SCPI Enabled** then click **OK**. Once checked, you must reboot to un-check this 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;  $I_{out} < 10\text{mA}$ ;  $R_{out} = 100\ \text{ohms}$

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,  $R_{in}=100k$  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".

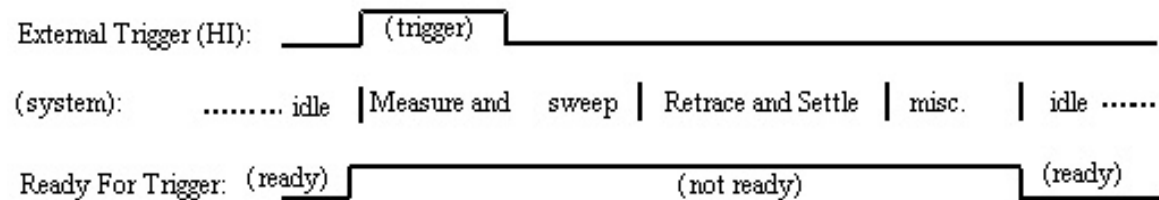
The preset state for Trigger is "Internal".

#### 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	TTL input - state may be read with SCPI or COM command
	Holdoff In	
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	<b>CAUTION:</b> 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.

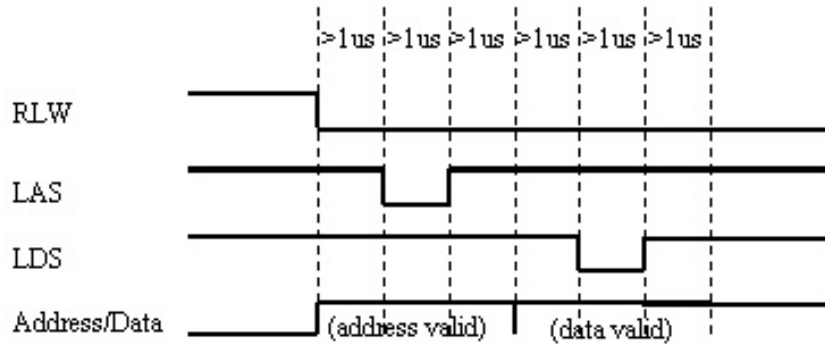
#### **HW Details**

Each of these I/O pins has a series 215-ohm resistor followed by 4.7k-ohm pull-up resistor.

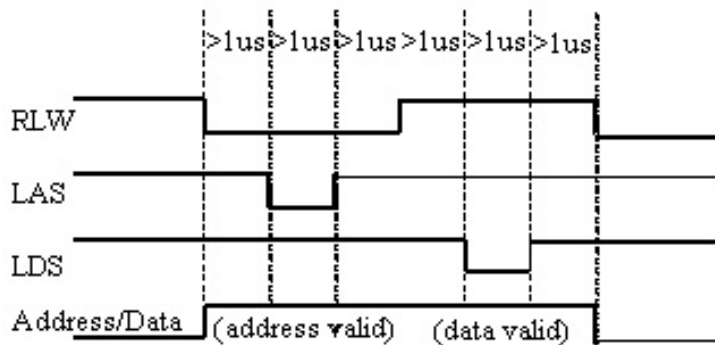
Write/Read is implemented by an output tri-state TTL buffer / latch for latching and enabling write data in parallel with a TTL input buffer for reading.

#### **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.

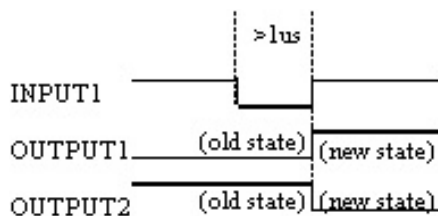
#### HW Details

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.

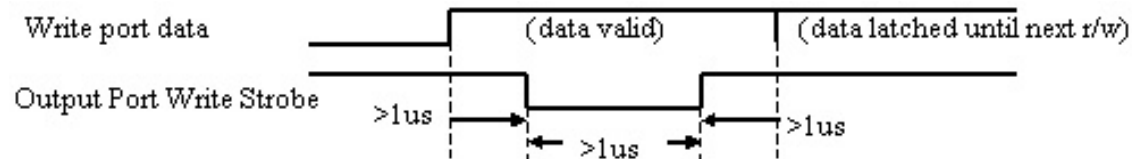
Connected to a TTL register output controlled by the analyzer.

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**

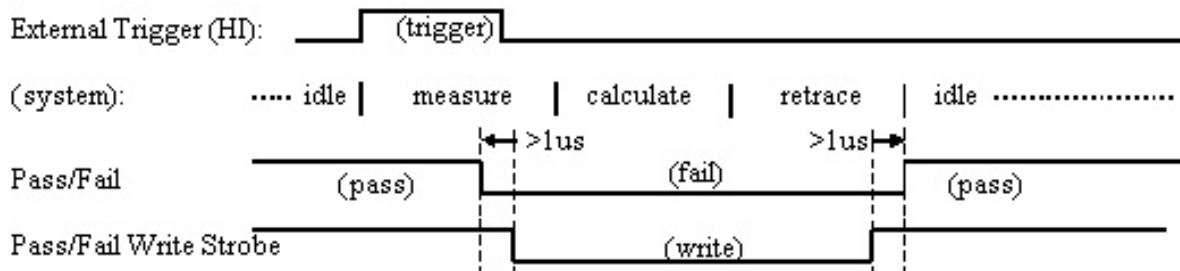
The Pass/Fail Out state is valid for at least 1us before Pass/Fail Write Strobe is pulled Low.

The Pass/Fail Out state is valid for at least 1us after Pass/Fail Write Strobe is pulled High.

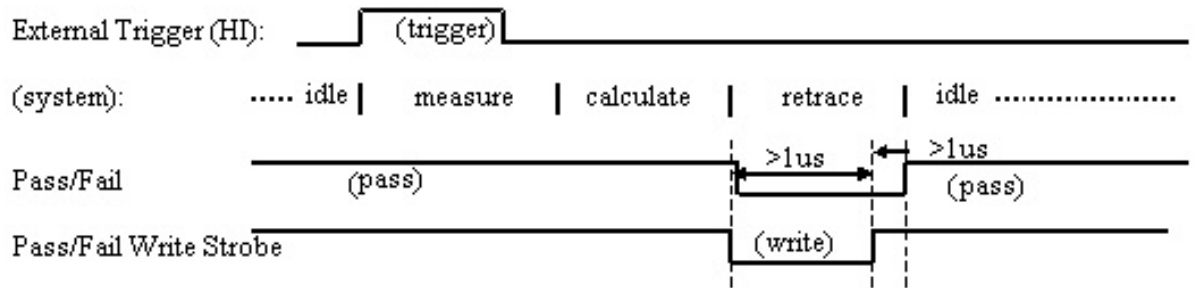
Pass/Fail Out is reset to its default state before the next measurement is started.

Pass/Fail Write Strobe will be Low for at least 1us.

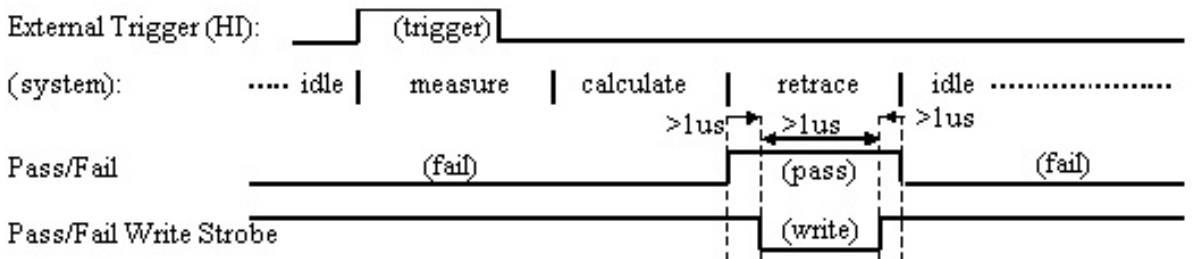
If the network analyzer is in **External Trigger** mode, Pass/Fail Write Strobe will go High (invalid) at least 1us before **Ready for Trigger** goes Low.



**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 (10 $\mu s$  minimum) indicates that the specified sweep event has finished. High output (10 $\mu s$  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.








## 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<DOSILIF> 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

ERC DONE  
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  
PWRCAL  
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	<011>><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	<011>><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 (Analog IN)	Integers 1-31	<011>><LF

**PNA SCPI Equivalent - Notes**

CONT:AUX:INP Reads the ADC input voltage from pin 14 of the AUX IO connector.

<b>PNA COM Equivalent - Notes</b>			
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	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>			
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.

<b>PNA COM Equivalent - Notes</b>			
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	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>			
SENS:SEGM			Turn on each segment to be used with list frequency sweep.

<b>PNA COM Equivalent - Notes</b>			
LimitSegment Object			LimitSegment object.

#### ATT

8753 Command	Description	Range	Query Response
ATTP1><num>[DB]	Selects the amount of attenuation at <b>PORT 1</b> .	0–70 dB	<num><LF
ATTP2><num>[DB]	Selects the amount of attenuation at <b>PORT 2</b> . Note: These commands only apply to 8753ES Option 011 analyzers.	0–70 dB	<num><LF

<b>PNA SCPI Equivalent - Notes</b>			
Step 1 SOUR:POW:COUP			Set Port Power Coupling OFF.
Step 2 SOUR:POW:ATT			Set the attenuation level for the selected port.

<b>PNA COM Equivalent - Notes</b>			
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

<b>PNA SCPI Equivalent - Notes</b>			
DISP:WIND:TRAC:Y:AUTO			Auto scale on the specified trace in the

specified window.

<b>PNA COM Equivalent - Notes</b>	
Autoscale Method	Auto scales the trace or all of the traces in the selected window.

**AVER**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
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<ONIOFF>	Turns averaging on and off on the active channel.	N/A	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>	
SENS:AVER:CLE	Restart averaging.
SENS:AVER:COUN	Read-Write the averaging factor.
SENS:AVER	Read-Write averaging ON or OFF.

<b>PNA COM Equivalent - Notes</b>	
Averaging Restart Method	Restart averaging.
Averaging Factor Property	Read-Write the averaging factor.
Averaging Property	Read-Write averaging ON or OFF.



**BLAD**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
BLAD<ONIOFF>	Blanks the display.	N/A	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>	
DISP:ENAB	Blanks the display information in <b>all</b> windows.
DISP:WIND:ENABLE	Blanks the display information in a <b>specified</b> window.

<b>PNA COM Equivalent - Notes</b>	
Visible Property	Makes the Network Analyzer application visible or not visible.

**BR**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
BR	Measures and displays B/R on the active channel.	N/A	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>	
Step 1 CALC:PAR:DEF Step 2 DISP:WIND	Follow the steps below to create and display a measurement. Create the measurement. 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<AIB>	Edits a specified 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

Access the appropriate table in either collection.

PowerLossSegments 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	<011><LF>
CALIRAI	Begins the sequence for a response and isolation calibration.	N/A	<011><LF>
CALIRESP	Begins the sequence for a response calibration.	N/A	<011><LF>
CALIS111	Begins the sequence	N/A	<011><LF>

	for an S11 1-port calibration (ES models), or a reflection 1-port calibration (ET models).		
CALIS221	Begins the sequence for an S22 1-port calibration.	N/A	<01><LF>
CALITRL2	Begins the sequence for a thru, reflect, line or line, reflect, match (TRL*/LRM*) 2-port calibration.	N/A	<01><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 <b>forward</b> enhanced response calibration.	N/A	<01><LF>
CALIRERC	Begins the sequence for a <b>reverse</b> enhanced response calibration.	N/A	<01><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	<01>><LF>
CALK292MM	Selects a 2.92-mm calibration kit as the default cal kit.	N/A	<01>><LF>
CALK292S	Selects a 2.92* calibration kit (85056K) as the default cal kit.	N/A	<01>><LF>
CALK35MD	Selects a 3.5-mm calibration kit (85052B/D for 8720E series analyzers, and	N/A	<01>><LF>

CALK35MC	85033D for 8753ET/ES analyzers) as the default cal kit. 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.	N/A	<01>><LF
CALK716	Selects a 7-16 calibration kit (85038) as the default cal kit.	N/A	<01>><LF
CALK7MM	Selects a 7-mm calibration kit (85050 series for 8720E series analyzers, and 85031B for 8753ET/ES analyzers) as the default cal kit.	N/A	<01>><LF
CALKN50	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.	N/A	<01>><LF
CALKN75	Selects a type-N 75 ohm calibration kit (85036B/E) as the default cal kit.	N/A	<01>><LF
CALKTRLK	Selects a TRL 3.5-mm calibration kit (85052C) as the default cal kit.	N/A	<01>><LF
CALKUSED	Selects a user-defined calibration kit.	N/A	<01>><LF

**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	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:COLL:METH REFL1		Read-Write the calibration method. Set to "NONE" to turn calibration off.	
<b>PNA COM Equivalent - Notes</b>			
CalibrationType_Property		Specifies and returns the type of calibration to be <b>applied</b> to the measurement. Set to "NONE" to turn calibration off.	

## CENT

8753 Command	Description	Range	Query Response
CENT<num>[HZIDB]	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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<num><LF
<b>PNA SCPI Equivalent - Notes</b>			
SENS:FREQ:CENT		To set the center frequency value.	
SENS:SEGM:FREQ:CENT		To set the center frequency value <b>for a segment</b> .	
<b>PNA COM Equivalent - Notes</b>			
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
<b>Notes</b>			

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	<01>><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	N/A	<01>><LF

trigger mode.

<b>PNA SCPI Equivalent - Notes</b>	
INIT:CONT	Read-Write the sweep triggering mode.
<b>PNA COM Equivalent - Notes</b>	
Continuous_Method	Read-Write the sweep triggering mode.

#### CORI

8753 Command	Description	Range	Query Response
CORI<ONIOFF>	Turns interpolative error correction on and off.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>		Read-Write correction interpolation ON or OFF.	
<b>PNA COM Equivalent - Notes</b>		No equivalent command at present.	

#### CORR

8753 Command	Description	Range	Query Response
CORR<ONIOFF>	Turns error correction on and off.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>		Read-Write whether or not correction data is applied to the measurement.	
<b>PNA COM Equivalent - Notes</b>		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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<num><LF
<b>PNA SCPI Equivalent - Notes</b>		Read-Write the Continuous Wave (or Fixed) frequency.	
<b>PNA COM Equivalent - Notes</b>		CW Frequency property.	

#### CWTIME

8753 Command	Description	Range	Query Response
CWTIME	Selects CW time as the sweep type.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			

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	<01>><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	<01>><LF
DELRFIXM	Makes the fixed marker the delta reference.	N/A	<01>><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**

8753 Command	Description	Range	Query Response
DELA	Displays the data formatted as group delay.	N/A	<01>><LF

**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**

8753 Command	Description	Range	Query Response
DISPDATA	Data only.	N/A	<011>><LF
DISPDATM	Data and memory.	N/A	<011>><LF
DISPDDM	Data divided by memory (linear division, log subtraction). See also "DIVI."	N/A	<011>><LF
DISPDMM	Data minus memory (linear subtraction). See also "MINU."	N/A	<011>><LF
DISPMEMO	Memory only.	N/A	<011>><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.

**DIVI**

8753 Command	Description	Range	Query Response
DIVI	Data divided by memory (linear division, log subtraction). See also "DISPDDM."	N/A	<011>><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.

**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	<011><LF>
<b>PNA SCPI Equivalent - Notes</b>			
		No equivalent command at this time	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
1. SENS:CORR:COLL:ACQ ECAL<AIB> 2. *OPC?		Measures the ECAL A module Operation Complete query	
<b>PNA COM Equivalent - Notes</b>			
		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	<011><LF>
<b>PNA SCPI Equivalent - Notes</b>			
1. SENS:CORR:COLL:METH SPARSOLT 2. SENS:CORR:COLL:ACQ ECAL<AIB>		Sets the calibration method to SOLT Measures the ECAL module	
<b>PNA COM Equivalent - Notes</b>			
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 ECAL isolation averages function	1-999	<num><LF>
<b>PNA SCPI Equivalent - Notes</b>			
SENS:AVER:COUN		Sets the number of measurement sweeps to combine for an average.	
<b>PNA COM Equivalent - Notes</b>			
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>
<b>PNA SCPI Equivalent - Notes</b>			
		No equivalent command at this time	
<b>PNA COM Equivalent - Notes</b>			
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	<011><LF>
<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:ISOL		Turns isolation cal ON or OFF during Full 2-port (or ECAL) calibration.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
1. SENS:CORR:COLL:METH REFL3 2. SENS:CORR:COLL:ACQ ECAL<AIB>		Sets the calibration method to 1-port Measures the ECAL module	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
1. SENS:CORR:COLL:METH REFL3 2. SENS:CORR:COLL:ACQ ECAL<AIB>		Sets the calibration method to 1-port Measures the ECAL module	
<b>PNA COM Equivalent - Notes</b>			
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	N/A	N/A

EDITLIML	power loss list. 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	N/A	<011>><LF



	data (real and imaginary pairs) along with the other files.		
EXTMDATO	Selected data arrays only (real and imaginary pairs), without instrument states or calibrations. Always saves the data array, even if it hasn't been selected.	N/A	<01>><LF
EXTMFORM	Formatted trace data. Uses currently selected format for data.	N/A	<01>><LF
EXTMRAW	Raw data arrays (real and imaginary pairs).	N/A	<01>><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:DATA:CUST		Read-Write either measurement data or memory data.	
<b>PNA COM Equivalent - Notes</b>			
Get_Data_Method		Retrieves data.	
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
EXTMGRAP	User graphics.	N/A	<01>><LF
<b>PNA SCPI Equivalent - Notes</b>			
		No equivalent command at present.	
<b>PNA COM Equivalent - Notes</b>			
PrintToFile_Method		Saves the screen data to bitmap (.bmp) file of the screen.	
<hr/>			
<b>EXTT</b>			
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
EXTT	Activates or deactivates the external trigger mode. OPC-compatible.	N/A	<01>><LF
EXTTPOIN	Sets the external trigger to auto-trigger on point. OPC-compatible.	N/A	<01>><LF
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
			No equivalent command at present.
<b>PNA COM Equivalent - Notes</b>			
			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 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.	N/A	N/A
FORM3	IEEE 64-bit floating-point format, 8 bytes-	N/A	N/A

	<p>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.</p>		
FORM4	<p>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.</p>	N/A	N/A
FORM5	<p>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 comply with PC-DOS formats. If you are using a PC-based controller, FORM5 is the most effective format to use.</p>	N/A	N/A

**PNA SCPI Equivalent - Notes**

format:data	FORM1, the 8753 analyzer's internal binary format, is not compatible with PNA. 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**

8753 Command	Description	Range	Query Response
FRER	Places the analyzer in GPIB free run mode. (Same as continuous sweep trigger mode.) See "CONT."	N/A	<01>><LF

**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 calibration sequence.	N/A	N/A

**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	<01>><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,xxxxxxxx ,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,	<num><LF

3000, 3700, 6000

<b>PNA SCPI Equivalent - Notes</b>	
SENS:BWID	Read-Write the bandwidth of the digital IF filter to be used in the measurement.

<b>PNA COM Equivalent - Notes</b>	
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	<011>><LF

<b>PNA SCPI Equivalent - Notes</b>	
CALC:FORM	Read-Write the display format for the measurement.

<b>PNA COM Equivalent - Notes</b>	
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

<b>PNA SCPI Equivalent - Notes</b>	
CALC:DATA:	Writes Measurement data, Memory data, or Error terms

<b>PNA COM Equivalent - Notes</b>	
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	N/A	N/A

INPURAW1	data array, using the current setting of the FORM command. 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
<b>PNA SCPI Equivalent - Notes</b>			
Step 1 CALC:DATA: Step 2 SENS:CORR		Input the data array. 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.	
<b>PNA COM Equivalent - Notes</b>			
Step 1aPut_Data_Complex_Method Step 1bPut_Data_Complex_Method Step 2 Put_Data_Complex_Method		Input raw data. Input formatted data 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	N/A	N/A

arrays for channels 1  
and 2 in FORM4 only

**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
INPUALK<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	Amplitude range.	Currently this command can be



	limit line segment.		queried by sending the command by the OUTPUTI 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.
<b>PNA SCPI Equivalent - Notes</b>			
CALC:LIM:DATA		Read-Write data for limit lines.	
<b>PNA COM Equivalent - Notes</b>			
LimitSegment_Object		Make a limit line object.	

## LIMI

-  LIMIAMPO
-  LIMILINE
-  LIMIMAOF
-  LIMISTIO
-  LIMITEST

8753 Command	Description	Range	Query Response
LIMILINE<ONIOFF>	Turns the display of the limit lines on and off.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:LIM:DISP:STAT		Read-Write the display of limit lines ON or OFF.	
<b>PNA COM Equivalent - Notes</b>			
LineDisplay_Property		Turns the display of limit lines ON or OFF.	
<b>PNA SCPI Equivalent - Notes</b>			
CALC:LIM:STAT		Read-Write limit line <b>testing</b> ON or OFF.	
<b>PNA COM Equivalent - Notes</b>			
State_Property		Turns an object ON and OFF.	

8753 Command	Description	Range	Query Response
LIMIAMP0<num>[HZ IDB]	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]	Enters the stimulus	Stimulus range.	<num>><LF

DB]                   offset of the limit lines.

**Notes**  
These commands currently are not available.

#### LIMT

8753 Command	Description	Range	Query Response
LIMTFL	Makes the segment a flat line.	N/A	<01>><LF
LIMTSL	Makes the segment a sloping line.	N/A	<01>><LF
LIMTSP	Makes the segment a single point.	N/A	<01>><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	<01>><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	<01>><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
- LISPW RM

8753 Command	Description	Range	Query Response
LISFREQ	Selects the list frequency sweep mode.	N/A	<01>><LF

#### PNA SCPI Equivalent - Notes

SENS:SWE:TYPE                   Selects the sweep type.

#### PNA COM Equivalent - Notes



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	<011>><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
MANTRIG	Sets the trigger mode to manual trigger on point. OPC-compatible.	N/A	<011>><LF

**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> | MARKMAXI |
| MARKBUCK        | MARKMIDD |
| MARKCENT        | MARKMINI |
| MARKCONT        | MARKOFF  |

 MARKCOUP  
 MARKCW  
 MARKDELA  
 MARKDISC  
 MARKFAUV  
 MARKFSTI  
 MARKFVAL

 MARKREF  
 MARKSPAN  
 MARKSTAR  
 MARKSTIM  
 MARKSTOP  
 MARKUNCO  
 MARKZERO

8753 Command	Description	Range	Query Response
MARK<1 2 3 4 5><num>	Makes the selected marker active and sets its stimulus value.	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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<num><LF

**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 <b>center</b> stimulus value to that of the active marker's stimulus value.	N/A	N/A
MARKSTAR	Sets the <b>start</b> stimulus to that of the active marker's.	N/A	N/A
MARKSTOP	Sets the <b>stop</b> stimulus to that of the active marker's.	N/A	N/A
MARKREF	Sets the <b>reference value</b> 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 <b>center</b> 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 <b>start</b> 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 <b>stop</b> 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 <b>reference level</b> to the marker's Y-axis value.

8753 Command	Description	Range	Query Response
MARKDELA	Sets electrical length so group delay is zero at the active marker's stimulus.	N/A	N/A

**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.
---------------------------	---

8753 Command	Description	Range	Query Response
MARKMAXI	Search for trace maximum on the current channel. Same as SEAMAX.	N/A	<011>><LF
MARKMINI	Search for trace minimum on the	N/A	<011>><LF

current channel.  
Same as SEAMIN.

**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.

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------

MARKCONT	Places the markers continuously on the trace, not on discrete points (interpolates the marker values between discrete points).	N/A	<011>><LF
MARKDISC	Places the markers on the discrete measurement points.	N/A	<011>><LF

**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.

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------

MARKOFF	Turns all markers and marker functions off.	N/A	<011>><LF
---------	---	-----	-----------

**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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<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 delta reference marker.	N/A	N/A
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	N/A	<011>><LF



MARKUNCO	between the channels, as opposed to MARKUNCO. Uncouples the markers between channels, as opposed to MARKCOUP.	N/A	<01>><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: $\pm 500$ dB. For phase: $\pm 500$ degrees. For Smith chart and Polar: $\pm 500$ units. For linear magnitude: $\pm 500$ units. For SWR: $\pm 500$ 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	<011>><LF
MEASB	Measures and displays input B on the active channel.	N/A	<011>><LF
MEASR	Measures and displays input R on the active channel.	N/A	<011>><LF

**PNA SCPI Equivalent - Notes**

Step 1 CALC:PAR:DEF  
 Step 2 DISP:WIND  
 Step 3 DISP:WIND:TRAC:FEED

Create the measurement.  
 If a new window will be used to display the measurement, then create a window.  
 Display the measurement in the window.

**PNA COM Equivalent - Notes**

CreateMeasurement\_Method  
 Create and display the measurement.

**MEASSTAT**

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------

MEASTAT<ON/OFF> Turns trace statistics on and off. N/A <011>><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	<011>><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	<011>><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.
<b>PNA COM Equivalent - Notes</b>	
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
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:ISOL		Read-Write isolation cal ON or OFF during Full 2-port calibration.	
<b>PNA COM Equivalent - Notes</b>			
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	<01>><LF>
<b>PNA SCPI Equivalent - Notes</b>			
*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	N/A	<\$><LF >

data is in FORM1.

<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:COLL:CKIT:NAME			Read-Write a name for the selected calibration kit.
<b>PNA COM Equivalent - Notes</b>			
Name CalKit Property			Sets and Returns a name for the selected calibration kit.
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
OUTPDATA	Outputs the error-corrected data from the active channel in real/imaginary pairs.	N/A	<array><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:DATA			Output the error-corrected data array.
<b>PNA COM Equivalent - Notes</b>			
GetNACComplex Method			Output the error-corrected data array.
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
OUTPFORM	Outputs the formatted display data array from the active channel, in current display units.	N/A	<array><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:DATA			Output the data array.
<b>PNA COM Equivalent - Notes</b>			
GetData Method			Output the data array.
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
OUTPIDEN	Outputs the identification string for the analyzer in the form: AGILENT TECHNOLOGIES ,87NNEX,xxxxxxxx ,X.XX where 87NNEX is the model number of the instrument, xxxxxxxx is the serial number of the instrument, and X.XX is the firmware revision of the instrument. (Same as the "IDN?" command.)	N/A	<\$><LF >
<b>PNA SCPI Equivalent - Notes</b>			
*IDN?			Returns a string that uniquely identifies the analyzer.
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
SOUR:POW:CORR:DATA		Writes and reads source power calibration data.	
<b>PNA COM Equivalent - Notes</b>			
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 1-1><LF
<b>PNA SCPI Equivalent - Notes</b>			
STAT:QUES:LIM1:COND?		Check status bit to determine status of the limit test.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
No equivalent command at present.			
<b>PNA COM Equivalent - Notes</b>			
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 >
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
Stimulus Property		Sets and reads the X-Axis value of the marker.	
Value Property		Reads the Y-Axis value of the marker.	
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
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
<b>PNA SCPI Equivalent - Notes</b>			
CALC:DATA		Read-Write either measurement data or memory data. When querying memory, you must first store a trace into memory using CALCuate<num>:MATH:MEMorize	
<b>PNA COM Equivalent - Notes</b>			
DataToMemory Method		If the data is not in memory, store data into memory.	
GetData Method		Output memory data.	
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
OUTPMSTA	Outputs the marker statistics in ASCII format: mean, standard deviation, 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.	N/A	<num,num,num><LF
<b>PNA SCPI Equivalent - Notes</b>			
Step 1 CALC:FUNC:TYPE		Select the statistic TYPE that you can then query.	
Step 2 CALC:FUNC:DATA?		Read the selected trace statistic.	
<b>PNA COM Equivalent - Notes</b>			
Get Trace Statistics Method		Returns the Trace Statistics.	
<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
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	N/A	<num,num,num><LF >

OUTPMWIL	turned off again. 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=S11 data 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	N/A	<\$><LF
----------	-----------------------	-----	---------



contains the serial number of the analyzer.

**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=S11 data 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	N/A	<array><LF

OUTPSEGF	value, min stimulus, min value for all active segments. This is an ASCII transfer. Outputs the limit test status for a specified segment. See also "SELSEG."	N/A	<011-1><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 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.	N/A	<array><LF
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,	N/A	<learnstring><LF

	and the list frequency table. It is always in binary format not intended for decoding.		
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=S11 data 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 information.	N/A	Rows of data separated by a <LF>. Ends with <LF><LF>.
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
<b>PNA SCPI Equivalent - Notes</b>			
CONTRol:AUXiliary:C:DATA		Reads and writes a 4-bit value to Port C on the Aux I/O connector.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
CALC:CORR:OFFS:PHAS		Read-Write the phase offset for the selected measurement.	
<b>PNA COM Equivalent - Notes</b>			
Phase Offset Property		Sets the Phase Offset.	

---

## PHAS

8753 Command	Description	Range	Query Response
PHAS	Selects the phase display format.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:FORM		Read-Write the display format for the measurement.	
<b>PNA COM Equivalent - Notes</b>			
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 segment.	Choose from: 3, 11, 21, 26, 51, 101, 201, 401, 801, 1601	<num><LF
<b>PNA SCPI Equivalent - Notes</b>			
SENS:SWE:POIN		Read-Write the number of data points for the measurement.	
<b>PNA COM Equivalent - Notes</b>			
Number of Points Property		Sets or returns the Number of Points.	

---

## POL

8753 Command	Description	Range	Query Response
POLA	Selects the polar display format.	N/A	<011>><LF
POLMLIN	Selects linear as the marker readout format for polar display.	N/A	<011>><LF
POLMLOG	Selects log as the marker readout format for polar display.	N/A	<011>><LF
POLMRI	Selects real/imaginary as the marker readout format for polar display.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:FORM		Selects the polar display format.	

CALC:MARK  
 CALC:MARK:FORM

Use this command to turn on a marker.  
 Selects the appropriate marker readout format.

**PNA COM Equivalent - Notes**

Format Property  
 Marker Format Property

Selects the polar display format.  
 Selects the appropriate marker readout format.

**PORE**

8753 Command	Description	Range	Query Response
PORE<ON/OFF>	Turns port extensions on and off.	N/A	<011>><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	<011>><LF
PORT2<num>[S]	Set the port extension length for Port 2	±10 seconds	<011>><LF
PORTA<num>[S]	Set the port extension length for Input A	±10 seconds	<011>><LF
PORTB<num>[S]	Set the port extension length for Input B	±10 seconds	<011>><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  
 Port2 Property  
 InputA Property  
 InputB Property

Sets the port extension value for Port 1.  
 Sets the port extension value for Port 2.  
 Sets the port extension value for Receiver A.  
 Sets the port extension value for Receiver B.

**PORTP**

8753 Command	Description	Range	Query Response
PORTP<CPLDI/UNCPLD>	Selects either coupled or uncoupled for the port powers of a given channel.	N/A	<011>><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
<b>PNA SCPI Equivalent - Notes</b>			
SOUR:POW		Read-Write the RF power output level.	
<b>PNA COM Equivalent - Notes</b>			
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 >
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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	Use two-digit integers 0007.	N/A

also "PRAN."

<b>PNA SCPI Equivalent - Notes</b>	
SOUR:POW:ATT	Setting the attenuation is equivalent to setting the source power range.
<b>PNA COM Equivalent - Notes</b>	
Attenuator Property	Setting the attenuation is equivalent to setting the source power range.

**POWS**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
POWS	Selects power sweep from the sweep type menu.	N/A	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>		Read-Write the type of analyzer sweep mode.	
SENS:SWE:TYPE			
<b>PNA COM Equivalent - Notes</b>		Sets the type of X-axis sweep that is performed on a channel.	
Sweep Type Property			

**POWT**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
POWT<ON/OFF>	Sets source power on or off. Works the opposite of the SOUP command. Sending POWTON turns source power off. Sending POWTOFF turns source power on.	N/A	<011>><LF>
<b>PNA SCPI Equivalent - Notes</b>		Turns ON and OFF Source Power.	
OUTP			
<b>PNA COM Equivalent - Notes</b>		Turns ON and OFF Source Power.	
Source Power State Property			

**PRAN**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
PRAN<num>	Sets the source power range. See also "POWR."	integers 07.	N/A
<b>PNA SCPI Equivalent - Notes</b>		Setting the attenuation is equivalent to setting the source power range.	
SOUR:POW:ATT			
<b>PNA COM Equivalent - Notes</b>		Setting the attenuation is equivalent to setting the source power range.	
Attenuator Property			

**PRES**

<b>8753 Command</b>	<b>Description</b>	<b>Range</b>	<b>Query Response</b>
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**

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------

PWMCONES	Power Meter Cal done on one sweep. A calibration sweep should be taken. a calibration sweep should be taken (TAK) to ensure a valid power calibration.	-100dB to 100dB	<01><<LF>
----------	--	-----------------	-----------

**PNA SCPI Equivalent - Notes**

Selects the source power calibration method.

**PNA COM Equivalent - Notes****PWRR**

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------

PWRR<PMANIPAUT O>	Selects whether the power range is in auto or manual mode.	N/A	<01><<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	<011>><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 response and isolation calibration.	N/A	N/A
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.  
SetCallInfo Method Specifies the type of calibration to **perform**.

---

**REAL**

8753 Command	Description	Range	Query Response
REAL	Sets the display format to real.	N/A	<011>><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: $\pm 500$ dB. For phase: $\pm 500$ degrees. For Smith chart and Polar: $\pm 500$ units. For linear magnitude: $\pm 500$ units. For SWR: $\pm 500$ 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
<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:COLL:SAVE		Write-only to calculate the correction data.	
<b>PNA COM Equivalent - Notes</b>			
Calculate Error Coefficients Method		Calculates the correction data.	

## REST

8753 Command	Description	Range	Query Response
REST	Measurement restart.	N/A	N/A
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
		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
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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.

SetCallInfo 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	<011>><LF
S12	Reverse transmission measurement.	N/A	<011>><LF
S21	Forward transmission measurement.	N/A	<011>><LF
S22	Reverse reflection measurement.	N/A	<011>><LF

### PNA SCPI Equivalent - Notes

Follow the steps below to create and display a measurement.  
Step 1 CALC:PAR:DEF  
Step 2 DISP:WIND  
Step 3 DISP:WIND:TRAC:FEED  
Create the measurement.  
If a new window will be used to display the measurement, then create a window.  
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: $\pm 500$ dB. For phase: $\pm 500$ degrees. For Smith chart and Polar: $\pm 500$ units. For linear magnitude: $\pm 500$ units. For SWR: $\pm 500$ units. The scale is always positive, and has minimum values of 0.001dB, 10e-12 degrees, 10e-15 seconds, and 10 picounits.	<num><LF
<b>PNA SCPI Equivalent - Notes</b>			
DISP:WIND:TRAC:Y:PDIV		Read-Write the Y axis <b>Per Division</b> value of the specified trace in the specified window.	
<b>PNA COM Equivalent - Notes</b>			
YScale Property		Sets or returns the Y-axis <b>Per Division</b> 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
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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.	
<b>8753 Command</b>			
<b>Description</b>	<b>Range</b>	<b>Query Response</b>	
SEAMAX	Search for trace maximum on the current channel.	N/A	<01><LF>
SEAMIN	Search for trace minimum on the current channel.	N/A	<01><LF>
<b>PNA SCPI Equivalent - Notes</b>			
CALC:MARK:FUNC:EXEC		Write-only to immediately execute (perform) the specified search function.	
<b>PNA COM Equivalent - Notes</b>			
Search Max Method		Searches the marker domain for the maximum value.	
Search Min Method		Searches the marker domain for the minimum value.	
<b>8753 Command</b>			
<b>Description</b>	<b>Range</b>	<b>Query Response</b>	
SEAOFF	Turns the marker search off.	N/A	<01><LF>
<b>PNA SCPI Equivalent - Notes</b>			
CALC:MARK		Turn marker search off by turning OFF the marker.	
<b>PNA COM Equivalent - Notes</b>			
Delete Marker Method		Turn marker search off by turning OFF the marker.	
<b>8753 Command</b>			
<b>Description</b>	<b>Range</b>	<b>Query Response</b>	
SEATARG<num>	Set the search target amplitude.	Amplitude range.	<num><LF>
<b>PNA SCPI Equivalent - Notes</b>			
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**

Sense:Segment  
Calc:Limit

PNA Network Analyzers allow one to directly edit a segment or limit line. To edit a segment or limit line, see the following commands:  
Commands to edit a segment.  
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



SENS:SEGM:POW  
segment.  
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 in 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	<011>><LF
SMIMGB	Selects G+jB (conductance and susceptance) marker readout on a Smith chart.	N/A	<011>><LF
SMIMLIN	Selects linear magnitude marker readout on a Smith chart.	N/A	<011>><LF
SMIMLOG	Selects log magnitude marker readout on a Smith chart.	N/A	<011>><LF
SMIMRI	Selects real/imaginary pairs (resistance and reactance) marker readout on a Smith chart.	N/A	<011>><LF
SMIMRX	Selects R + jX marker readout on a Smith chart.	N/A	<011>><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
SMOOPER<num>	Sets the smoothing aperture as a percent of the trace.	0.05 to 20%	<num><LF
SMOOO<ONIOFF>	Selects whether smoothing is on or off.	N/A	<011>><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<ONIOFF>	Selects whether the source power is on or off.	N/A	<011>><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>[HZIDB]	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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time}$	<num><LF

step.

**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>[HZIDB]	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: $\pm 1/\text{frequency step}$ . For CW time sweep, transform on: $\pm 1/\text{time step}$ .	<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
<b>PNA SCPI Equivalent - Notes</b>			
MMEM:STOR:STAT		Write-only to store the specified file.	
<b>PNA COM Equivalent - Notes</b>			
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 and frequency span.	N/A	N/A
SWET<num>[S]	Sets the sweep time. (Setting SWET0 is equivalent to sending the SWEA command.)	086,400 s	<num><LF
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
CALC:FORM		Read-Write the display format for the measurement.	
<b>PNA COM Equivalent - Notes</b>			
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	<011>><LF
<b>PNA SCPI Equivalent - Notes</b>			
		No equivalent command at present.	
<b>PNA COM Equivalent - Notes</b>			
GPIBMode Property		Selects the talker listener mode.	

---

## TAK

8753 Command	Description	Range	Query Response
TAKRS	Begins a receiver calibration sweep	N/A	N/A
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
DISP:WIND:TITL:DATA		Read-Write data in the window title area.	
<b>PNA COM Equivalent - Notes</b>			
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<ONIOFF>	Turns marker search tracking on and off.	N/A	<011>><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  
SENS:CORR:COLL  
Read-Write the calibration method.  
Write-only to measure the specified standard from the selected calibration kit.

#### PNA COM Equivalent - Notes

SetCalInfo\_Method  
AcquireCalStandard2 Method  
Specifies the type of calibration to **perform**.  
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>
<b>PNA SCPI Equivalent - Notes</b>			
*TST?		Returns the result of a complete self-test. An ASCII 0 indicates no failures found.	
<b>PNA COM Equivalent - Notes</b>			
		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
<b>PNA SCPI Equivalent - Notes</b>			
SENS:SWE:SRCP		Read-Write the source port when making non S-parameter measurements. Has no effect on S-parameter measurements.	
<b>PNA COM Equivalent - Notes</b>			
CreateMeasurement Method		Create and display the measurement. Method parameter allows one to select the specific port.	

---

**TTL**

8753 Command	Description	Range	Query Response
TTLPULS	TTL normally high, low pulse at end of sweep.	N/A	<0 1><L F >
TTLHPULS	TTL normally low, high pulse at end of sweep.	N/A	<0 1><L F >
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
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.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
SENS:CORR:RVEL:COAX		Read-Write the velocity factor to be used with Electrical Delay and Port Extensions.	
<b>PNA COM Equivalent - Notes</b>			
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
<b>PNA SCPI Equivalent - Notes</b>			
*WAI		Prohibits the instrument from executing any new commands until all pending overlapped commands have been completed.	
<b>PNA COM Equivalent - Notes</b>			
No equivalent command at present.			

---

**WID**

8753 Command	Description	Range	Query Response
--------------	-------------	-------	----------------



**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.

---