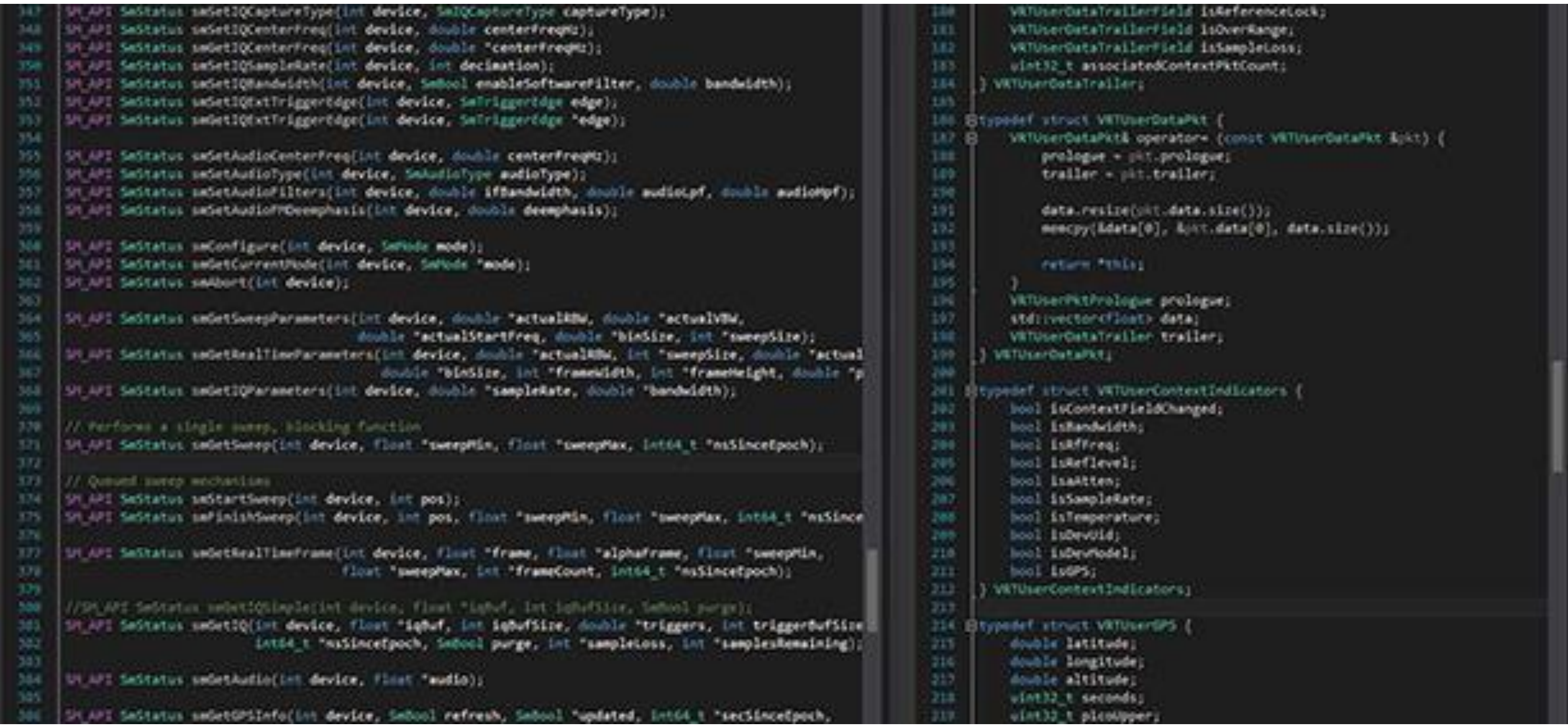

VSG60 SCPI Programming Manual

User Guide

VSG60 SCPI Programming Manual

Published 10/23/2020

1502 SE Commerce Ave, Suite 101
Battle Ground, WA
Phone 360-313-7997

Contents

1 Version Notes

SCPI commands can and will change as the VSG60 software evolves. It is recommended that when you update the VSG60 software in an installation that is controlled via SCPI, to review the version notes and determine if any functionality needs to be updated. See the separate document title *scpi_version_notes.txt* for a full list of changes for each version of the VSG60 software.

2 Introduction / About SCPI

SCPI (Standard Commands for Programmable Instruments) is a standard which covers the set of commands used to program various instruments. The standard covers the syntax, form, behavior, etc. of these commands in attempt to reduce development time for the user.

For the purposes of Signal Hound and the VSG60 software, a user can send SCPI commands to control the VSG60 software in an automatic fashion. SCPI commands are sent to instruments over many interfaces, commonly GPIB, VXI, USB, Ethernet, etc. The VSG60 software accepts commands over a network socket. The VSG60 software will accept a single network connection in which it can receive SCPI commands and send responses.

This document will cover the basics of SCPI commands, how to get started programming the VSG60 software, and will cover the full SCPI command set implemented by the VSG60 software.

3 SCPI command basics

This section contains a quick overview of the SCPI command syntax and usage to the extent that is relevant to the VSG60 software. The VSG60 does not utilize all functionality in the SCPI standard and as such said functionality will not be covered here.

3.1 Commands

A SCPI command is comprised of a series of keywords separated by colons. A command may be followed by a ‘?’ to represent a query, a series of parameters separated by spaces, or both.

`:SENSE:FREQUENCY:CENTER 1GHz` (Example command for setting the center frequency to 1GHz)

`:sense:frequency:center?` (Example command for querying the current center frequency)

Commands are case insensitive. Each keyword in a command can have a short and long form. Both can be used interchangeably.

`:SENSe:FREQuency:CENTer` is a command with three keywords. Each keyword has a short and long form. The short form is denoted by the uppercase characters and the long form is the full keyword including the upper and lower-case characters. For example, FREQ is the short form of FREQUENCY. When constructing a command, the short and long form can be interchanged.

For example, you could construct the command as such, `:SENS:FREQUENCY:CENT` where SENSE and CENTER are sent as short form and FREQUENCY as longform.

Some commands are options and are denoted as such by the '[]' characters.

`[:SENSe]:FREQuency:CENTer` is a command where the first keyword is optional. This command can be sent as `FREQ:CENT` and still be interpreted correctly.

Commands are terminated with a newline character. For example

```
:SENS:FREQ:CENT 1GHZ\n
```

Commands will be processed once a newline is reached. Additionally, a newline will reset the current keyword path.

3.2 Multiple commands

Multiple commands can be sent to the device at once using the semi colon character separating each command.

```
:SENS:FREQ:CENT 1GHz; :SENS:FREQ:SPAN 10MHz\n
```

This is an example of sending two commands at once. Additionally, when sending multiple commands, you don't need to repeat all keywords leading up to the final keyword for commands after the first.

```
:SENS:FREQ:CENT 1GHz; SPAN 10MHz\n
```

Here `SPAN` retains the `:SENS:FREQ:` keywords from the previous command. To prevent this from happening use the colon character leading the second command. For example

```
:SENS:FREQ:CENT 1GHz; :SPAN 10MHz\n
```

This is an invalid series of commands, since span is prefixed with a colon command which reset the previous keywords.

3.3 Parameters

There are several types of parameters that can be sent in commands.

Boolean	ON | OFF | 0 | 1
Keyword <bool>	Character specific strings for a given command. These keywords can also have short and long form.

Numeric
<integer>
<double>
	Numeric parameters take either the form of integer or decimal values. Examples include
	`1`
	`1.23`
	`9`

	`3.14`

Frequency
<freq>
	These are numeric parameters with a frequency suffix. Possible frequency suffixes include
	`HZ | KHZ | MHZ | GHZ`
	The suffixes are case insensitive. If a suffix is not present, Hz is the default unit. Examples include
	`1kHz`
	`20MHz`
	`12GHz`
	Any function that returns a frequency will return the frequency in Hz with no suffix present.

Amplitude
<amplitude>
	These are numeric parameters with an amplitude suffix. Possible amplitude suffixes include
	`DBM | DBMV | DBUV | MV`
	The suffixes are case insensitive. A suffix must be present unless indicated otherwise. Examples include
	`-20DBM`
	`60dbuv`
	If a function returns an amplitude, it will return the amplitude in the current software units without a suffix.

3.4 Return Values

Values returned from the VSG60 software (as a result of sending a query command) are separated by a semi-colon if multiple query commands are sent in one string and are terminated by a newline. For example, sending

```
“CALC:MARK:MAX; X?; Y?\n”
```

results in a return string of

```
“1000000;-20\n”
```

The command sent performs a peak search and queries the X and Y positions of the marker. The return is the X and Y positions separated by a semicolon and terminated with a newline.

3.5 Special Characters

This section describes the numerous special characters that are present in the commands in this document.

Character	Description	Example
|	Vertical stroke between parameters indicates multiple choices	`FLATtop | GAUSsian` The choices are between FLATTOP or GAUSSIAN. Provide one or the other.

[ ]	Square brackets indicate an optional keyword	`:SYSTem:ERRor[:NEXT]?`
		Next is an optional keyword and the command could also be composed as
		`:SYSTem:ERRor?`

<>	Angle brackets around a parameter indicate a type and angle brackets should not be included in the user command.	`*RCL <int>`
		<int> is the type of parameter and an example of using this command would be
		`*RCL 1`
		Notice the angle brackets are not included.

4 Getting Started

See the SCPI examples found in the SDK download on any of the Signal Hound product download pages. The examples use the C programming language and a common VISA library implementation.

Instrument control is performed by connecting to the VSG60 software on TCP/IP port 5024. On this port, a user can send and receive raw SCPI commands. It is not necessary to use a I/O library like VISA to communicate with the VSG60 software but it can simplify several operations. It is possible to communicate directly over the socket with socket programming. The computer that is communicating with the VSG60 software does not have to be the same computer running the VSG60 software and does not have to be a Windows platform.

It is recommended to use a VISA library if available. Several implementations of VISA exist. Commonly used ones include Keysight’s I/O libraries, and NI’s VISA libraries. You can also use VISA implementations that exist in other languages/environments such as MATLAB, LabVIEW, and Python.

Connecting to the socket interface using VISA looks like this

```
viOpen(rm, “TCPIP::localhost::5025:SOCKET”, VI_NULL, VI_NULL, &inst);
```

Additionally, when using a VISA library, it is necessary to set the VI_ATTR_TERMCHAR_EN attribute to true. This will terminate the read operation when the termination character is received. The termination character should be set to the newline (‘\n’) character if it is not set by default. The code for this is below.

```
viSetAttribute(inst, VI_ATTR_TERMCHAR_EN, VI_TRUE);

viSetAttribute(inst, VI_ATTR_TERMCHAR, '\n');
```

Only one connection to the VSG60 software can be active at a time. The connection can be terminated by either closing the socket connection, either through the socket library you are using, the viClose function if you are using a VISA library, or by closing your application. The VSG60 will immediately begin waiting for another socket connection when the previous one is ended.

5 Functionality provided through SCPI

The table below details what functionality is covered under the current SCPI command set. Functionality will be added over time. If functionality you need it not available, please contact us at aj@signalhound.com to make requests.

Functionality	Implemented
Amplitude Modulation	Yes
Frequency Modulation	Yes
Multitone	Yes
Step Sweep	Yes
Ramp Sweep	Yes
AWGN	Yes
Digital Mod	Yes
Bluetooth LE	No
IEEE 802.11 a/n/ac	No
Arb	Yes
Streaming	No

6 Examples

All SCPI examples are provided in the API SDK download which can be downloaded on any of the device download pages on the Signal Hound website.

7 Functions

7.1 Common Commands

The software supports the following common commands.

Command	`*IDN?`

	`*RCL <int>`
	`*SAV <int>`
	`*RST`
	`*TRG`
	`*OPC`
	`*ESR?`

Description	`*IDN?`, Query the serial number and name of the device. `*RCL`, Load preset [1-9]. `*SAV`, Save preset [1-9]. `*RST`, Same as `PRESet`, see below. `*TRG,` Triggers the device. `*OPC,` Tells the instrument that after all the commands are executed and finished to set the ESR bit 0 (OPC bit) to 1. This command in combination with the *ESR? command can be used for synchronization through polling. See the C++ SCPI examples in the SDK for an example of polling using these commands. `*ESR?,` Returns the Event Status Register (ESR). Only bit 0 is used at this time. Bit 0 represents Operation Complete (OPC). Returns 0 if *OPC has been seen but there are still commands to be executed and finished. Sends a 1 when all commands have been finished and executed. This command in combination with the *ESR? command can be used for synchronization through polling. See the C++ SCPI examples in the SDK for an example of polling using these commands.

Examples	`*IDN?`
	`*RCL 1`
	`*SAV 1`
	`*TRG`
	`*RST`
	`*OPC`
	`*ESR?`

Software Controls	Status Bar File Menu -> Presets -> Load File Menu -> Presets -> Save Preset Key Trigger Key
Couplings	None
Preset	N/A
Notes	

7.2 System Functions

The following commands are used to perform system level software actions and query information about the system.

Command	`:SYSTem:COMMunicate:GTLocal`
	`:SYSTem:CLOSe`
	`:SYSTem:PRESet`
	`:SYSTem:PRESet?`
	`:SYSTem:VERsion?`

Description	`COMMunicate:GTLocal`, Puts the software in local mode. `CLOSe`, Disconnect any active device and closes the software. There is not a way to reopen the software using SCPI commands. This will also terminate the socket connection. `PRESet`, Presets the active device. This will power cycled the active device and return the software to the initial power on state. This process can take between 6-20 seconds depending on the device type. `PRESet?`, Presets the active device. This will close and reopen the active device. This process can take between 6-20 seconds depending on the device type. Returns 0 or 1 depending on success. (1 for success) `VERsion?`, Returns the software version number.

Examples	`SYST:CLOS`
	`SYST:PRESET?`
	`SYSTEM:VERSION?`
	`SYST:COMM:GTL`

Software Controls	Status Bar File Menu -> File -> Exit Preset File Menu -> Help -> About Spike Remote Mode Dialog -> Return to Local
Couplings	None
Preset	N/A
Notes	

7.2.1 Device Management

The functions below allow you to remotely manage the active device in the software. This is useful for error recovery in the event a device disconnect occurs due, or if one is managing multiple Signal Hound devices on one PC.

Connecting Signal Hound devices can take between 3-20 seconds depending on the type of device and the state of the device prior to interfacing it. If the VISA timeout is shorter than the time it takes to connect the device in the software, you will need to loop on timeout until you receive the connect status return.

Command	`:SYSTem:DEVice:ACTive?`
	`:SYSTem:DEVice:COUNt?`
	`:SYSTem:DEVice:LIST?`
	`:SYSTem:DEVice:CONnect? <int>`
	`:SYSTem:DEVice:DISConnect?`

Description	`ACTive?`, Returns whether or not a device is currently connected and active in the software. Look at the `*IDN?` function to request information about the device. `COUNt?`, Returns the number of devices connected to the PC. No device may be active when this function is called. IE, you must call `DISConnect?` before calling this function.

	`LIST?`, Returns all serial numbers available. The serial numbers are returned as ascii integers and are comma separated. To determine how many serial numbers are present, use the `COUNt?` function. `CONnect?`, Connect a device. You need to provide the serial number of the device to connect. Returns 0 or 1 depending on if the device successfully opened. `DISConnect?`, Disconnects the active device. Returns 1 when finished.

Examples	`SYST:DEV:ACT?`
	`SYST:DEV:COUNT?`
	`SYSTEM:DEVICE:LIST?`
	`SYSTEM:DEVICE:CONNECT? 30700189`
	`SYSTEM:DEV:CONNECT?`
	`SYST:DEV:DISC?`

Software Controls	File Menu -> File -> Connect File Menu -> File -> Disconnect
Couplings	Only one device can be active at a time.
Preset	N/A
Notes	

7.2.2 Errors

The VSG60 software maintains a list of system errors available to the user. Errors are stored with a unique ID, name, and description. The types of issues represented in the error list are settings conflicts, SCPI issues such as invalid parameter types or instructions, file I/O errors, etc.

It is recommended to frequently check for errors when utilizing SCPI in the software. Check the SCPI examples to see how to quickly poll for any present errors.

The errors are returned in the form

```
"ID,description;error information"
```

ID is a unique integer for the error. The description is an ascii text description for the error, and error information is any additional context information for the error generated. An example error message is below.

```
"-2,Invalid Parameter;Expected frequency parameter"
```

This error indicates the SCPI parser was expecting a frequency parameter and was either unable to find it or was unable to parse it as a frequency.

Once the error queue is empty, the software will return the 'no error' error when the next system error is requested. 'No error' has an ID of 0.

Command	`:SYSTem:ERRor:COUNt?`
	`:SYSTem:ERRor[:NEXT]?`

	`:SYSTem:ERRor:CLEAr`
Description	`COUNt?`, Returns the number of errors in the error queue. `NEXT?`, Returns the next error in the queue, and removing it from the queue. `CLEAR`, Remove all errors from the queue, returns nothing.

Examples	`SYST:ERR:COUN?`
	`SYSTEM:ERROR:NEXT?`
	`SYST:ERR?`
	`SYST:ERR:CLEAR`

Software Controls	Utilities -> Show Error Log Error Info -> Clear Button
Couplings	None
Preset	N/A
Notes	None

7.3 Reference

These commands control the reference oscillator settings the of the spectrum analyzer.

Command	`[:SENSe]:ROSCillator:SOURce INTernal|EXTernal`
	`[:SENSe]:ROSCillator:SOURce?`

Description	Specify whether the generator should use the internal reference or use an external reference.

Examples	`:SENSE:ROSCILLATOR:SOURCE INTERNAL`
	`ROSC:SOUR EXT`
	`rosc:source?`

Software Controls	Ext Ref
Couplings	None
Preset	On program startup, internal reference is selected.
Notes	None

7.4 Output

Command	`:OUTPut[:STATe] ON|OFF|0|1`
	`:OUTPut[:STATe]?`
	`:OUTPut:MODulation[:STATe] ON|OFF|0|1`
	`:OUTPut:MODulation[:STATe]?`

Description	

Examples	`:OUTPUT 1`
	`:OUTPUT:MOD ON`

Software Controls	RF On/Off Mod On/Off
Couplings	None
Preset	On program startup, both RF and Mod off
Notes	None

7.5 Frequency

Command	`[:SOURce]:FREQuency <freq>`
	`[:SOURce]:FREQuency?`
	`[:SOURce]:FREQuency:STEP[:INCRement] <freq>`
	`[:SOURce]:FREQuency:STEP[:INCRement]?`

Description	

Examples	`FREQ 2.45GHz`
	`FREQ?`
	`FREQ:STEP 20MHz`
	`FREQ:STEP?`

Software Controls	Freq Step
Couplings	None
Preset	
Notes	None

7.6 Power

Command	`[:SOURce]:POWer <double>`
	`[:SOURce]:POWer?`
	`[:SOURce]:POWer:STEP[:INCRement] <double>`
	`[:SOURce]:POWer:STEP[:INCRement]?`

Description	

Examples	`POW -20`
	`POW?`
	`POW:STEP 1`
	`POW:STEP?`

Software Controls	Level (dBm) Step (dB)
Couplings	None
Preset	
Notes	None

7.7 Impairments

Command	`:OUTPut:IMPairments:FREQuency:OFFSet <freq>`
	`:OUTPut:IMPairments:FREQuency:OFFSet?`
	`:OUTPut:IMPairments:LSPur[:STATe] <bool>`
	`:OUTPut:IMPairments:LSPur[:STATe]?`
	`:OUTPut:IMPairments:IOFFset <int>`
	`:OUTPut:IMPairments:IOFFset?`
	`:OUTPut:IMPairments:QOFFset <int>`
	`:OUTPut:IMPairments:QOFFset?`
	`:OUTPut:IMPairments:SRATe:MULTiplier <double>`
	`:OUTPut:IMPairments:SRATe:MULTiplier?`
	`:OUTPut:IMPairments:AWGN[:STATe] <bool>`

	`:OUTPut:IMPairments:AQGN[:STATe]?`
	`:OUTPut:IMPairments:AWGN:SNR <double>`
	`:OUTPut:IMPairments:AWGN:SNR?`
	`:OUTPut:IMPairments:AWGN:IBWidth <freq>`
	`:OUTPut:IMPairments:AWGN:IBWidth?`

Description	

Examples	`OUTP:IMP:FREQ:OFFS 1MHz`
	`OUTP:IMP:LSP ON`
	`OUTP:IMP:IOFF 10`
	`OUTP:IMP:QOFF -22`
	`OUTP:IMP:SRAT:MULT 1`
	`OUTP:IMP:AWGN ON`
	`OUTP:IMP:AWGN:SNR 30`
	`OUTP:IMP:AWGN:IBW 10MHz`

Software Controls	Impairment Controls -> Frequency Offset Impairment Controls -> Low Spur Mode Impairment Controls -> I Offset Impairment Controls -> Q Offset Impairment Controls ->Sample Rate Error (ppm) Impairment Controls -> AWGN Enabled Impairment Controls -> AWGN SNR (dB) Impairment Controls -> AWGN Bandwidth
Couplings	None
Preset	
Notes	None

7.8 Amplitude Modulation

Command	`[:SOURce]:AM[:STATe] <bool>`
	`[:SOURce]:AM[:STATe]?`
	`[:SOURce]:AM:FREQuency <freq>`
	`[:SOURce]:AM:FREQuency?`
	`[:SOURce]:AM:SHAPe SINE|TRIangle|SQUare|RAMP`
	`[:SOURce]:AM:SHAPe?`
	`[:SOURce]:AM:DEPTh[:LINear] <double>`
	`[:SOURce]:AM:DEPTh[:LINear]?`

Description	

Examples	`AM ON`
	`AM:FREQ 10kHz`
	`AM:SHAPE SINE`
	`AM:DEPTH 50`

Software Controls	AM Controls -> Enabled AM Controls -> Rate AM Controls -> Depth(%) AM Controls -> Shape
Couplings	None
Preset	

Notes	None

7.9 Frequency Modulation

Command	`[:SOURce]:FM[:STATe] <bool>`
	`[:SOURce]:FM[:STATe]?`
	`[:SOURce]:FM:FREQuency <freq>`
	`[:SOURce]:FM:FREQuency?`
	`[:SOURce]:FM:SHAPe SINE|TRIangle|SQUare|RAMP`
	`[:SOURce]:FM:SHAPe?`
	`[:SOURce]:FM:DEViation <double>`
	`[:SOURce]:FM:DEViation?`

Description	

Examples	`FM ON`
	`FM:FREQ 20kHz`
	`FM:SHAPE RAMP`
	`FM:DEV 100kHz`

Software Controls	FM Controls -> Enabled FM Controls -> Rate FM Controls -> Deviation FM Controls -> Shape
Couplings	None
Preset	
Notes	None

7.10 Pulse Modulation

Command	`[:SOURce]:PULM[:STATe] <bool>`
	`[:SOURce]:PULM[:STATe]?`
	`[:SOURce]:PULM:TRIGger:TYPE SINGle|CONTinuous`
	`[:SOURce]:PULM:TRIGger:TYPE?`
	`[:SOURce]:PULM:INTernal:PWIDth <time>`
	`[:SOURce]:PULM:INTernal:PWIDth?`
	`[:SOURce]:PULM:INTernal:PERiod <time>`
	`[:SOURce]:PULM:INTernal:PERiod?`

Description	

Examples	`PULM ON`
	`PULM:TRIG:TYPE CONT`
	`PULM:INT:PWID 10us`
	`PULM:INT:PER 1ms`

Software Controls	Pulse Controls -> Enabled Pulse Controls -> Trigger Mode Pulse Controls -> Width Pulse Controls -> Period
Couplings	None
Preset	
Notes	None

7.11 Multitone

<table>
<tr><td>Command</td><td>[:SOURce]:MTONe[:STATe] <bool>
[:SOURce]:MTONe[:STATe]?
[:SOURce]:MTONe:PHASe FIXed|RANDom|PARAbolic
[:SOURce]:MTONe:PHASe?
[:SOURce]:MTONe:PHASe:SEED <int>
[:SOURce]:MTONe:PHASe:SEED?
[:SOURce]:MTONe:NTONes <int>
[:SOURce]:MTONe:NTONes?
[:SOURce]:MTONe:FSPacing <freq>
[:SOURce]:MTONe:FSPacing?
[:SOURce]:MTONe:FNOTch <freq>
[:SOURce]:MTONe:FNOTch?</td></tr>
<tr><td>Description</td><td></td></tr>
<tr><td>Examples</td><td>MTON ON
MTON:PHAS PARA
MTON:PHAS:SEED 1234
MTON:NTON 1001
MTON:FSP 10kHz
MTON:FNOT 1MHz</td></tr>
<tr><td>Software Controls</td><td>Multitone Controls -> Enabled
Multitone Controls -> Tone Phase
Multitone Controls -> Seed
Multitone Controls -> Tone Count
Multitone Controls -> Freq Spacing
Multitone Controls -> Notch Width</td></tr>
<tr><td>Couplings</td><td>None</td></tr>
<tr><td>Preset</td><td></td></tr>
<tr><td>Notes</td><td>None</td></tr>
</table>

7.12 Step Sweep

<table>
<tr><td>Command</td><td>[:SOURce]:STEP[:STATe] <bool>
[:SOURce]:STEP[:STATe]?
[:SOURce]:STEP:TRIGger:TYPE SINGle|CONTinuous
[:SOURce]:STEP:TRIGger:TYPE?
[:SOURce]:STEP:TYPE FREQ|FREQAMPL
[:SOURce]:STEP:TYPE?
[:SOURce]:STEP:FREQuency:STARt <freq>
[:SOURce]:STEP:FREQuency:STARt?
[:SOURce]:STEP:FREQuency:STOP <freq>
[:SOURce]:STEP:FREQuency:STOP?
[:SOURce]:STEP:POINts <int>
[:SOURce]:STEP:POINts?
[:SOURce]:STEP:AMPLitude:STARt <double></td></tr>
</table>

	`[:SOURce]:STEP:AMPLitude:STARt?`
	`[:SOURce]:STEP:AMPLitude:STOP <double>`
	`[:SOURce]:STEP:AMPLitude:STOP?`
	`[:SOURce]:STEP:DWELl <time>`
	`[:SOURce]:STEP:DWELl?`

Description	

Examples	`STEP ON`
	`STEP:TRIG:TYPE SING`
	`STEP:TYPE FREQ`
	`STEP:FREQ:STAR 1GHz`
	`STEP:FREQ:STOP 2GHz`
	`STEP:POIN 1000`
	`STEP:AMPL:START -20`
	`STEP:AMPL:STOP -100`
	`STEP:DWEL 100ms`

Software Controls	Step Sweep Controls -> Enabled Step Sweep Controls -> Trigger Mode Step Sweep Controls -> Sweep Type Step Sweep Controls -> Start Freq Step Sweep Controls -> Stop Freq Step Sweep Controls -> Points Step Sweep Controls -> Start Level Step Sweep Controls -> Stop Level Step Sweep Controls -> Dwell Time
Couplings	None
Preset	
Notes	None

7.13 Ramp Sweep

Command	`[:SOURce]:RAMP[:STATe] <bool>`
	`[:SOURce]:RAMP[:STATe]?`
	`[:SOURce]:RAMP:TRIGger:TYPE SINGle|CONTinuous`
	`[:SOURce]:RAMP:TRIGger:TYPE?`
	`[:SOURce]:RAMP:FREQuency:SPAN <freq>`
	`[:SOURce]:RAMP:FREQuency:SPAN?`
	`[:SOURce]:RAMP:SWEep:TIME <time>`
	`[:SOURce]:RAMP:SWEep:TIME?`
	`[:SOURce]:RAMP:SWEep:PERiod <time>`
	`[:SOURce]:RAMP:SWEep:PERiod?`

Description	

Examples	`RAMP ON`
	`RAMP:TRIG:TYPE SING`
	`RAMP:FREQ:SPAN 20MHz`
	`RAMP:SWE:TIME 1ms`
	`RAMP:SWE:PER 1s`

Software Controls	Ramp Sweep Controls -> Enabled Ramp Sweep Controls -> Trigger Mode

	Ramp Sweep Controls -> Span Ramp Sweep Controls -> Sweep Time Ramp Sweep Controls -> Period
Couplings	None
Preset	
Notes	None

7.14 AWGN

Command	`[:SOURce]:RADio:AWGN[:STATe] <bool>`
	`[:SOURce]:RADio:AWGN[:STATe]?`
	`[:SOURce]:RADio:AWGN:BWIDth <freq>`
	`[:SOURce]:RADio:AWGN:BWIDth?`
	`[:SOURce]:RADio:AWGN:LENgth <time>`
	`[:SOURce]:RADio:AWGN:LENgth?`
	`[:SOURce]:RADio:AWGN:SEED <int>`
	`[:SOURce]:RADio:AWGN:SEED?`

Description	

Examples	`RAD:AWGN ON`
	`RAD:AWGN:BWID 20M`
	`RAD:AWGN:LEN 100ms`
	`RAD:AWGN:SEED 23`

Software Controls	AWGN Controls -> Enabled AWGN Controls -> Bandwidth AWGN Controls -> Length AWGN Controls -> Seed
Couplings	None
Preset	
Notes	None

7.15 Custom Digital Modulation

Command	`[:SOURce]:RADio:CUSTom[:STATe] <bool>`
	`[:SOURce]:RADio:CUSTom[:STATe]?`
	`[:SOURce]:RADio:CUSTom:TRIGger:TYPE SINGle|CONTinuous`
	`[:SOURce]:RADio:CUSTom:TRIGger:TYPE?`
	`[:SOURce]:RADio:CUSTom:SRATe <freq>`
	`[:SOURce]:RADio:CUSTom:SRATe?`
	`[:SOURce]:RADio:CUSTom:MODulation[:TYPE] BPSK|DBPSK|QPSK|DQPSK|OQPSK|P4DQPSK|PSK8|D8PSK|PSK16|QAM16|QAM64|QAM256|QAM1024|ASK|FSK2|FSK4|FSK8|FSK16|CUSTom`
	`[:SOURce]:RADio:CUSTom:MODulation[:TYPE]?`
	`[:SOURce]:RADio:CUSTom:FILTer RNYQuist|NYQuist|GAUSsian|RECTangle|CUSTom`
	`[:SOURce]:RADio:CUSTom:FILTer?`
	`[:SOURce]:RADio:CUSTom:FILTer:ALPHa <double>`
	`[:SOURce]:RADio:CUSTom:FILTer:ALPHa?`

```
[:SOURce]:RADio:CUSTom:FILTer:LENgth <int>
[:SOURce]:RADio:CUSTom:FILTer:LENgth?
[:SOURce]:RADio:CUSTom:DATA PN7|PN9|PN15|PN21|CUSTom
[:SOURce]:RADio:CUSTom:DATA?
[:SOURce]:RADio:CUSTom:DATA:SEED <int>
[:SOURce]:RADio:CUSTom:DATA:SEED?
[:SOURce]:RADio:CUSTom:MODulation:FSK[:DEViation] <freq>
[:SOURce]:RADio:CUSTom:MODulation:FSK[:DEViation]?
[:SOURce]:RADio:CUSTom:OVERsample <int>
[:SOURce]:RADio:CUSTom:OVERsample?
```

Description

Examples

```
RAD:CUST ON
RAD:CUST:TRIG:TYPE SING
RAD:CUST:SRAT 1MHz
RAD:CUST:MOD QAM16
RAD:CUST:FILT RNYQ
RAD:CUST:FILT:ALPH 0.2
RAD:CUST:FILT:LEN 16
RAD:CUST:DATA PN15
RAD:CUST:DATA:SEED 11
RAD:CUST:MOD:FSK:DEV 250kHz
RAD:CUST:OVER 4
```

Software Controls

Digital Mod Controls -> Enabled
Digital Mod Controls -> Trigger Mode
Digital Mod Controls -> Symbol Rate
Digital Mod Controls -> Modulation Type
Digital Mod Controls -> Filter Type
Digital Mod Controls -> Filter Alpha
Digital Mod Controls -> Filter Length (symbols)
Digital Mod Controls -> Sequence
Digital Mod Controls -> Sequence Seed
Digital Mod Controls -> FSK Deviation
Digital Mod Controls -> Oversample

Couplings: None

Preset

Notes: None

7.16 Arb

Command

```
[:SOURce]:RADio:ARB[:STATe] <bool>
[:SOURce]:RADio:ARB[:STATe]?
[:SOURce]:RADio:ARB:TRIGger:TYPE SINGle|CONTinuous
[:SOURce]:RADio:ARB:TRIGger:TYPE?
[:SOURce]:RADio:ARB:SRATe <freq>
[:SOURce]:RADio:ARB:SRATe?
[:SOURce]:RADio:ARB:IQ:SCALe:AUTO[:STATe] <bool>
[:SOURce]:RADio:ARB:IQ:SCALe:AUTO[:STATe]?
[:SOURce]:RADio:ARB:IQ:SCALe <double>
```

	`[:SOURce]:RADio:ARB:IQ:SCALe?`
	`[:SOURce]:RADio:ARB:IQ:SCALe:AVERage[:STATe] <bool>`
	`[:SOURce]:RADio:ARB:IQ:SCALe:AVERage[:STATe]?`
	`[:SOURce]:RADio:ARB:SAMPle:PERiod <int>`
	`[:SOURce]:RADio:ARB:SAMPle:PERiod?`
	`[:SOURce]:RADio:ARB:SAMPle:OFFSet <int>`
	`[:SOURce]:RADio:ARB:SAMPle:OFFSet?`
	`[:SOURce]:RADio:ARB:SAMPle:COUNt <int>`
	`[:SOURce]:RADio:ARB:SAMPle:COUNt?`
	`[:SOURce]:RADio:ARB:WAVeform?`
	`[:SOURce]:RADio:ARB:WAVeform:LENgth?`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD:CSV <filename>`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD:BINSC <filename>`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD:BINFC <filename>`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD:MIDAS <filename>`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD:IQ:ASCII <I1>, <Q1>, <I2>, <Q2>, …, <In>, <Qn>`
	`[:SOURce]:RADio:ARB:WAVeform:LOAD?`
	`[:SOURce]:RADio:ARB:WAVeform:UNLOAD`

Description	`STATe`, Enable/disable the Arb output mode. `TRIGger:TYPE`, Set the trigger mode for Arb output. `SRATe`, Set the Arb output sample rate. `IQ:SCALe:AUTO:STATe`, Enable/disable auto I/Q scaling. `IQ:SCALe`, Set the I/Q scale to be used when auto scaling is disabled. `IQ:SCALe:AVERage:STATe`, Enable/disable how to calculate the output power of the signal. `SAMPle:PERiod`, Set the waveform period in samples. Period is calculated after accounting for the offset and count. `SAMPle:OFFSet`, Set the waveform offset in samples. Specifies how many samples into the loaded waveform to start playback. Between offset and count, this allows users to only play a portion of the loaded waveform. `SAMPle:COUNt`, Specify the number of samples after the offset to output. Between offset and count, this allows users to only play a portion of the loaded waveform. `WAVeform?`, Queries the name of the loaded waveform. Returns an empty string is no file is loaded. `WAVeform:LENgth?`, Returns the total number of samples in the loaded waveform. The number returned does not include the offset and count values specified above. If no file is loaded, this returns 0. `LOAD`, Loads various file types. The file name provided must specify a file that matches the file type specified by the load SCPI function used. See the software UI manual for more information. `LOAD:BINSC`, Loads 16-bit complex integer binary file with provided filename. `LOAD:BINFC`, Loads 32-bit complex float binary file with provided filename. `LOAD:IQ:ASCII`, Load an I/Q waveform sent over SCPI. The I/Q values should be provided as alternating I/Q complex values, each I and Q value sent as a separate SCPI parameter, as ascii. A comma should separate all I/Q values. A comma should not be placed after the last Q value. An error will be thrown if an odd number of parameters is provided. See example below and programming example for usage.

	`LOAD?`, Returns 1 if a waveform is loaded. `UNLOAD`, Unloads any loaded waveform.

Examples	`RAD:ARB ON`
	`RAD:ARB:TRIG:TYPE SING`
	`RAD:ARB:SRAT 10MHz`
	`RAD:ARB:IQ:SCALE:AUTO ON`
	`RAD:ARB:IQ:SCALE 50`
	`RAD:ARB:IQ:SCALE:AVERAGE OFF`
	`RAD:ARB:SAMPLE:PERIOD 10000`
	`RADIO:ARB:SAMPLE:OFFSET 1024`
	`RAD:ARB:SAMP:OFFS?`
	`RAD:ARB:SAMP:COUNT 5000`
	`RAD:ARB:WAV?`
	`RAD:ARB:WAVEFORM:LENGTH?`
	
	`RAD:ARB:WAV:LOAD:CSV “file.csv”`
	Please note, that the quotations must appear in the command. If using a programming language like C/C++, you must escape sequence the quote in the string, for example
	`“RAD:ARB:WAV:LOAD:CSV \”file.csv\””`
	
	`RAD:ARB:WAV:LOAD:BINFC “file.bin”`
	`RAD:ARB:WAV:LOAD:IQ:ASCII 1.0, 0.0, 1.0, 0.0, -1.0, 0,0, -1,0, 0,0`
	<This line loaded an I/Q waveform with 4 I/Q samples where the first two samples were (1.0,0.0) and the last two samples were (-1.0, 0.0)> See the programming examples for another example of using this function.
	`RAD:ARB:WAV:LOAD?`
	`RAD:ARB:WAV:UNLOAD`

Software Controls	Arb Controls -> Enabled Arb Controls -> Trigger Mode Arb Controls -> Sample Rate Arb Controls -> Auto Scale Arb Controls -> I/Q Scale (%) Arb Controls -> Output Signal Average Arb Controls -> Period Arb Controls -> Sample Offset Arb Controls -> Samples to Use Arb Controls -> Samples in File Arb Controls -> Load Arb Controls -> Unload File
Couplings	None
Preset	No file loaded.
Notes	None