SDG6000X
Arbitrary Waveform Generator

Programming Guide

EN06A

SIGLENT TECHNOLOGIES CO., LTD.

Content

1 Programming Overview

By using USB and LAN interfaces, in combination with NI-VISA and programming languages, users can remotely control the waveform generator. Through the LAN interface, VXI-11, Sockets, and Telnet protocols can be used to communicate with the instruments. This chapter introduces how to build communication between the instrument and the PC. It also introduces how to configure a system for remote instrument control.

1.1 Build communication via VISA

1.1.1 Install NI-VISA

Before programming, please make sure that you have properly installed the latest version of National Instruments NI-VISA Software.

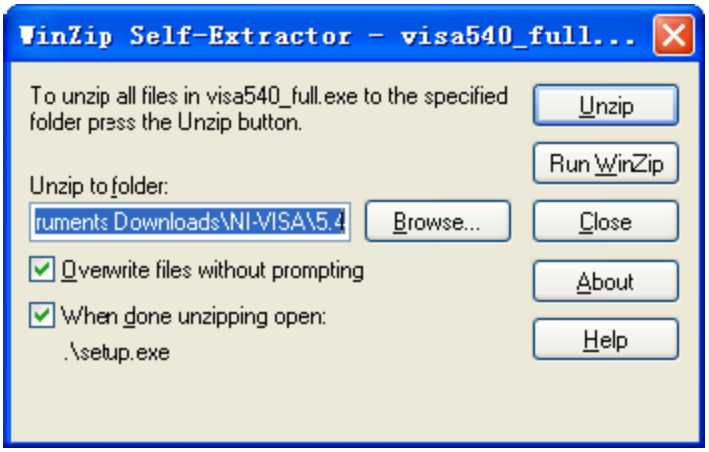
NI-VISA is a communication library that enables computer communications to instrumentation. There are two available VISA packages: A full version and the Run-Time Engine. The full version includes NI device drivers and a tool named NI MAX; a user interface to control the device. While the drivers and NI MAX can be useful, they are not required for remote control. The Run-Time Engine is a much smaller file and is recommanded for remote control.

For convenience, you can obtain the latest version of the NI-VISA run-time engine or the full version from the National Instruments website. The installation process is similar for both versions.

Follow these steps to install NI-VISA (The full version of NI-VISA 5.4 is used in this example):

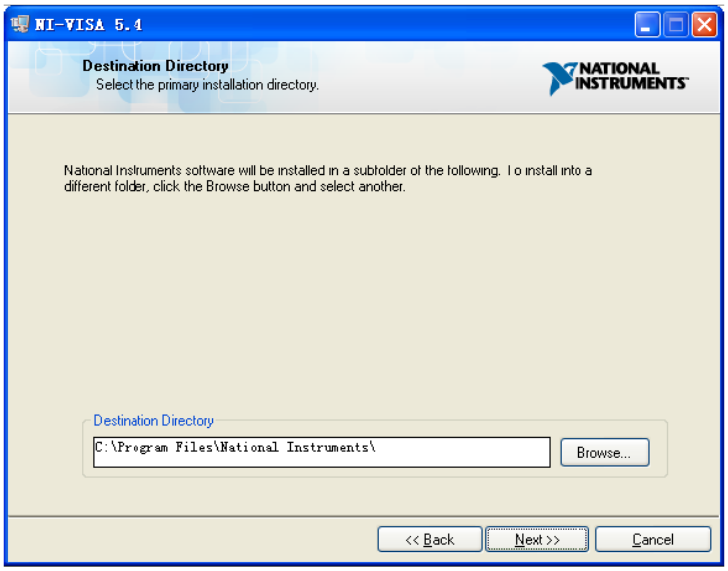
a. Download the appropriate version of NI-VISA (the Run-time engine is recommanded)

b. Double click the visa540_full.exe and observe the dialog box as shown below:

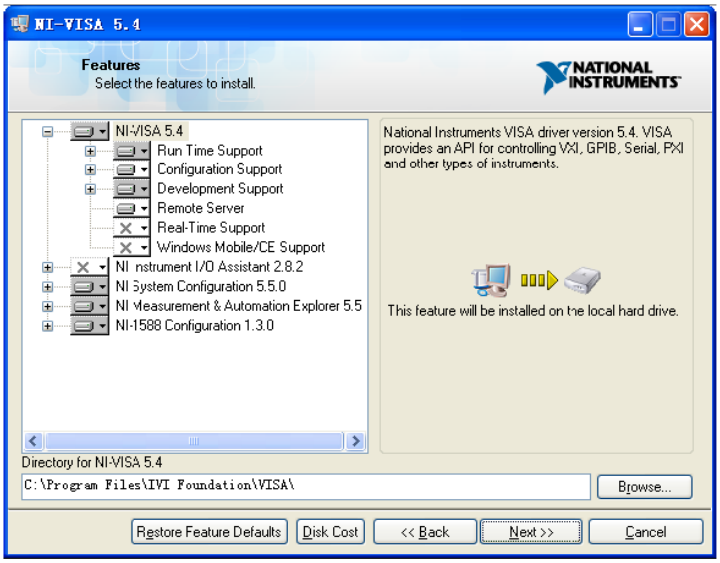


c. Click Unzip, the install process will launch after unzipping files. If your computer needs to install the .NET Framework 4, it may auto-start.

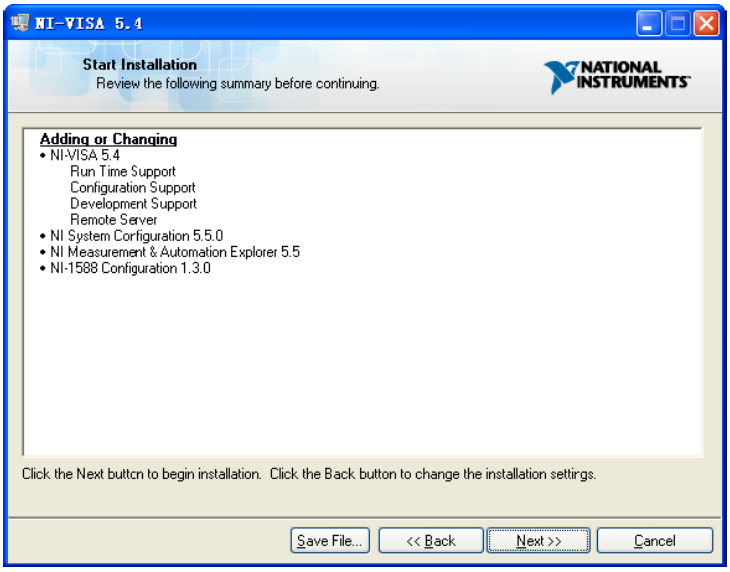
d. The NI-VISA install dialog is shown above. Click Next to start the installation process.



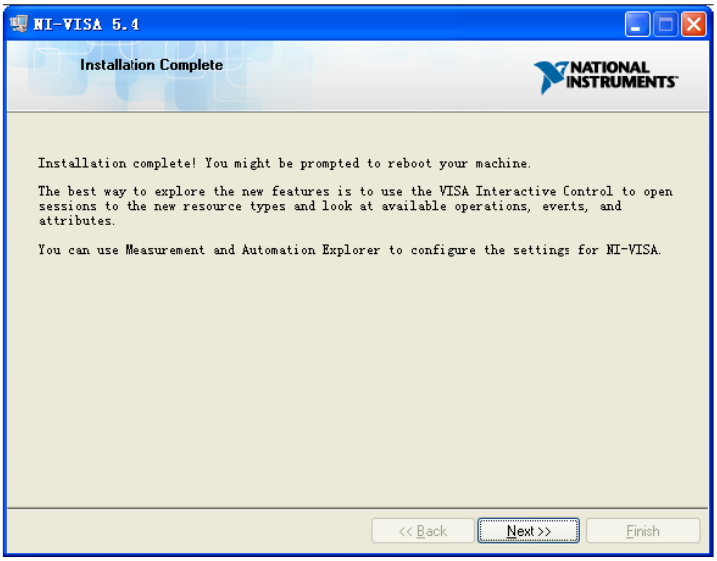
e. Set the install path, the default path is “C:\Program Files\National Instruments\”, you can change it if you prefer. Click Next, dialog as shown above.



f. Click Next twice, in the License Agreement dialog, select the “I accept the above 2 License Agreement(s).”, and click Next, and a dialog box will appear as shown below:



g. Click Next to begin installation.

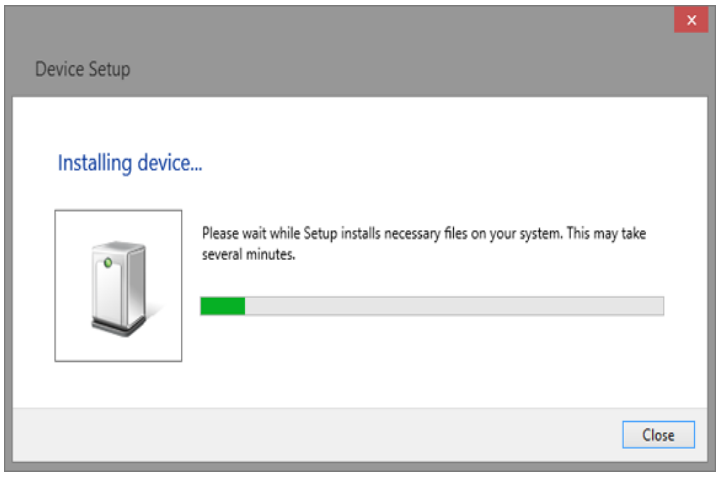


h. Now the installation is complete. Reboot your PC.

1.1.2 Connect the instrument

Depending on the specific model, the arbitrary waveform generator may be able to communicate with a PC through the USB or LAN interface.

Connect the arbitrary waveform generator and the USB Host interface of the PC using a USB cable. Assuming your PC is already turned on, turn on the SDG, and then the PC will display the “Device Setup” screen as it automatically installs the device driver as shown below.



Wait for the installation to complete and then proceed to the next step.

1.2 Remote Control

1.2.1 User-defined Programming

Users can send SCPI commands via a computer to program and control the arbitrary waveform generator. For details, refer to the introductions in "Programming Examples Programming Examples".

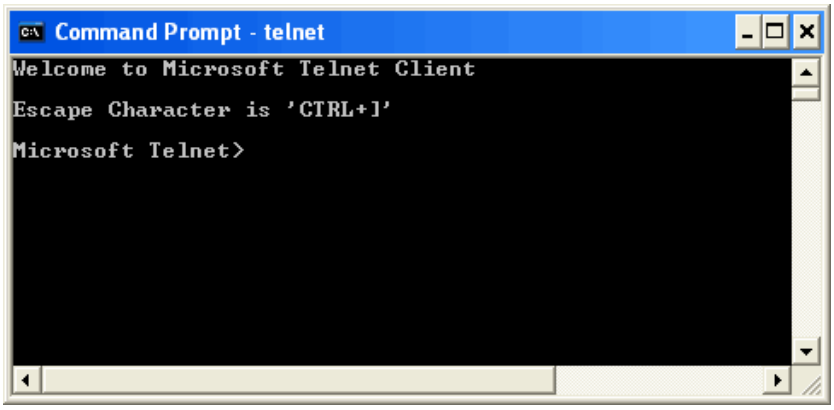
1.2.2 Using SCPI via NI-MAX

NI-MAX is a program created and maintained by National Instruments. It provides a basic remote control interface for VXI, LAN, USB, GPIB, and Serial communications. The SDG can be controlled remotely by sending SCPI commands via NI-MAX.

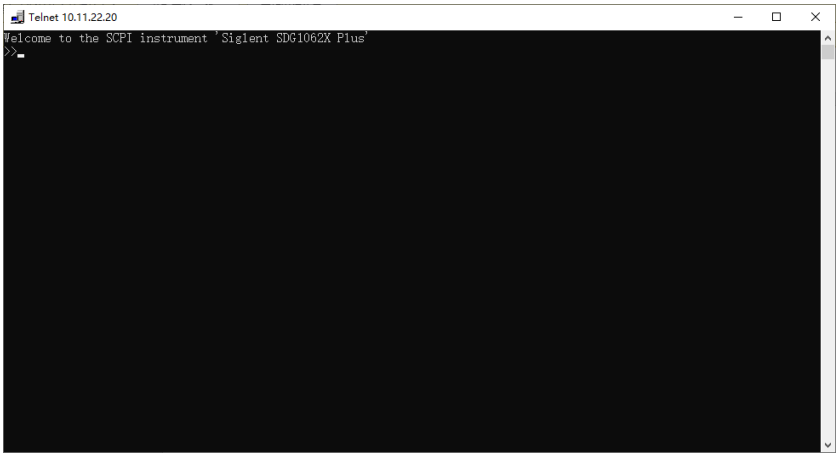
1.2.3 Using SCPI over Telnet

Telnet provides a means of communicating with the SDG over the LAN. The Telnet protocol sends SCPI commands to the SDG from a PC and is similar to communicating with the SDG over USB. It sends and receives information interactively: one command at a time. The Windows operating systems use a command prompt style interface for the Telnet client. The steps are as follows:

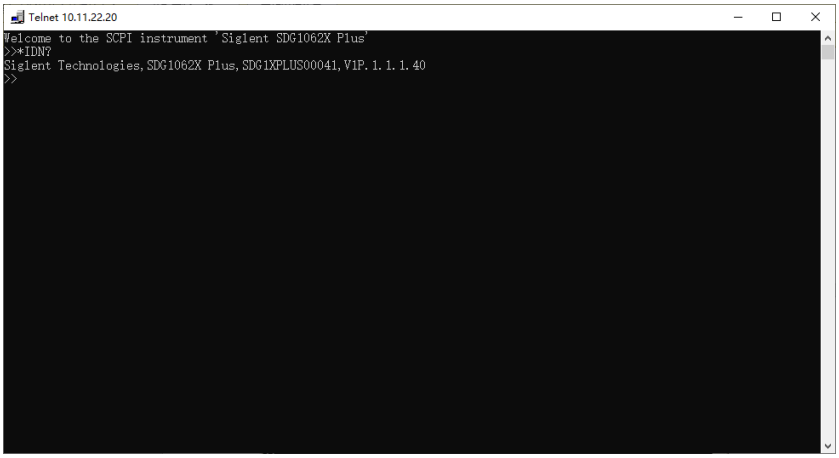
1. On your PC, click Start > All Programs > Accessories > Command Prompt.
2. At the command prompt, type in telnet.
3. Press the Enter key. The Telnet display screen will be displayed.



4. At the Telnet command line, type:open XXX.XXX.XXX.XXX 5024
   Where XXX.XXX.XXX.XXX is the instrument's IP address and 5024 is the port. You should see a response similar to the following:



5. At the SCPI> prompt, input the SCPI commands such as *IDN? to return the company name, model number, serial number, and firmware version number.



6. To exit the SCPI> session, press the Ctrl+] keys simultaneously.
7. Type quit at the prompt or close the Telnet window to close the connection to the instrument and exit Telnet.

1.2.4 Using SCPI over Socket

Socket API can be used to control the SDG series by LAN without installing any other libraries. This can reduce the complexity of programming.

SOCKET ADDRESS	IP address + port number
IP ADDRESS	SDG IP address
PORT NUMBER	5025

Please see section 5.2 "Examples of Using Sockets" for the details.

2 Introduction to the SCPI Language

2.1 About Commands & Queries

This section lists and describes the remote control commands and queries recognized by the instrument. All commands and queries can be executed in either the local or remote state.

Each command or query, with syntax and other information, has some examples listed. The commands are given in both long and short format at "**COMMAND SYNTAX**" and "**QUERY SYNTAX**", and the subject is indicated as a command or query or both. Queries perform actions such as obtaining information from the instrument and are identified by a question mark (?) following the header.

2.2 Description

In the description, a brief explanation of the function performed is given. This is followed by a presentation of the formal syntax, with the header given in Upper-and-Lower-Case characters and the short form derived from it in ALL UPPER-CASE characters. Where applicable, the syntax of the query is given with the format of its response.

2.3 Usage

The commands and queries listed here can be used for SDGxxxx Series Arbitrary Waveform Generators.

2.4 Command Notation

The following notations are used in the commands:

< > Angular brackets enclose words that are used as placeholders, of which there are two types: the header path and the data parameter of a command.

:= A colon followed by an equals sign separates a placeholder, from the description of the type and range of values that may be used in a command instead of the placeholder.

{ } Braces enclose a list of choices, one of which must be made.

[ ] Square brackets enclose optional items.

…] Ellipsis (trailing dots) indicate that the preceding element may be repeated one or more times.

2.5 Table of Command & Queries

Short	Long Form	Subsystem	What Command/Query does
*IDN	*IDN	SYSTEM	Gets identification from device.
*OPC	*OPC	SYSTEM	Gets or sets the OPC bit (0) in the Event Status Register (ESR).
*RST	*RST	SYSTEM	Restore default settings
CHDR	COMM_HEADER	SIGNAL	Sets or gets the command returned format
OUTP	OUTPUT	SIGNAL	Sets or gets the output state.
BSWV	BASIC_WAVE	SIGNAL	Sets or gets the basic wave parameters.
MDWV	MODULATEWAVE	SIGNAL	Sets or gets the modulation parameters.
SWWV	SWEEPWAVE	SIGNAL	Sets or gets the sweep parameters.
BTWV	BURSTWAVE	SIGNAL	Sets or gets the burst parameters.
PACP	PARACOPY	SIGNAL	Copies parameters from one channel to the other.
ARWV	ARBWAVE	DATA	Changes arbitrary wave type.
SYNC	SYNC	SIGNAL	Sets or gets the synchronization signal.
NBFM	NUMBER_FORMAT	SYSTEM	Sets or gets the data format.
LAGG	LANGUAGE	SYSTEM	Sets or gets the language.
SCFG	SYS_CFG	SYSTEM	Sets or gets the power-on system setting way.
BUZZ	BUZZER	SYSTEM	Sets or gets the buzzer state.
SCSV	SCREEN_SAVE	SYSTEM	Sets or gets the screen save state.
ROSC	ROSCILLATOR	SIGNAL	Sets or gets the state of the clock source.
FCNT	FREQCOUNTER	SIGNAL	Sets or gets the frequency counter parameters.

Short	Long Form	Subsystem	What Command/Query does
INVT	INVERT	SIGNAL	Sets or gets the polarity of the current channel.
COUP	COUPLING	SIGNAL	Sets or gets the coupling parameters.
VOLTPRT	VOLTPRT	SYSTEM	Sets or gets the state of over-voltage protection.
STL	STORELIST	SIGNAL	Lists all stored waveforms.
WVDT	WVDT	SIGNAL	Sets and gets the arbitrary wave data.
VKEY	VIRTUALKEY	SYSTEM	Sets the virtual keys.
SYST:COMM:LAN:IPAD	SYSTEM:COMMUNICATE:LAN:IPADDRESS	SYSTEM	The Command can set and get the system IP address.
SYST:COMM:LAN:SMAS	SYSTEM:COMMUNICATE:LAN:SMASK	SYSTEM	The Command can set and get the system subnet mask.
SYST:COMM:LAN:GAT	SYSTEM:COMMUNICATE:LAN:GATEWAY	SYSTEM	The Command can set and get the system Gateway.
HARM	HARMonic	SIGNAL	Sets or gets the harmonic information.
CMBN	CoMBiNe	SIGNAL	Sets or gets the wave combine information.
MODE	MODE	SIGNAL	Sets or gets the waveform phase mode
CASCADE	CASCADE	SYSTEM	Set up multi-device synchronization

3 Commands and Queries

3.1 IEEE 488.2 Common Command Introduction

The IEEE standard defines the common commands used for querying the basic information of the instrument or executing basic operations. These commands usually start with "*" and the length of the keywords of the command is usually 3 characters.

3.1.1 *IDN

DESCRIPTION	The *IDN? query causes the instrument to identify itself. The response is comprised of the manufacturer, model, serial number, and firmware version.
QUERY SYNTAX	*IDN?
RESPONSE FORMAT	Format 1: *IDN, <device id>,<model>,<serial number>,<firmware version>, <hardware version> Format 2: <manufacturer>,<model>,<serial number>,<firmware version> <device id>:= “SDG”. <manufacturer>:= “Siglent Technologies”. <model>:= A model identifier less than 14 characters, should not contain the word “MODEL”. <serial number>:= The serial number. <firmware version>:= The firmware version number. <hardware version>:= The hardware level field, containing information about all separately revisable subsystems.
EXAMPLE	Reads version information: **IDN?* Return: *Siglent Technologies,SDG6022X,SDG08CEF472570, 6.01.01.38R3* (It may differ from each version)

Notes:

1. Format of <hardware version>: value1-value2-value3-value4-value5.
   value1: PCB version.
   value2: Hardware version.
   value3: Hardware subversion.
   value4: FPGA version.
   value5: CPLD version.

3.1.2 *OPC

DESCRIPTION	The *OPC (Operation Complete) command sets the OPC bit (bit 0) in the standard Event Status Register (ESR). This command has no other effect on the operation of the device because the instrument starts parsing a command or query only after it has completely processed the previous command or query. The *OPC? query always responds with the ASCII character 1 because the device only responds to the query when the previous command has been entirely executed.
COMMAND SYNTAX	*OPC
QUERY SYNTAX	*OPC?
RESPONSE FORMAT	1

3.1.3 *RST

DESCRIPTION	The *RST command initiates a device reset and recalls the default setup.
COMMAND SYNTAX	*RST
EXAMPLE	This example resets the signal generator to the default setup: **RST*

3.2 Output Command

DESCRIPTION	This command enables or disables the output port(s) at the front panel. The query returns “ON” or “OFF” and “LOAD”, “PLRT”, and “RATIO” parameters.
COMMAND SYNTAX	<channel>:OUTPut ON|OFF,LOAD,<load>,PLRT, <polarity> <channel>:= {C1, C2}. <load>:= {see the note below}. The unit is ohms. <polarity>:= {NOR, INVT}, in which NOR refers to normal, and INVT refers to invert.
QUERY SYNTAX	<channel>:OUTPut?
RESPONSE FORMAT	<channel>:OUTP ON|OFF,LOAD,<load>,PLRT, <polarity>
EXAMPLE	Turn on CH1: *C1:OUTP ON* Read CH1 output state: *C1:OUTP?* Return: *C1:OUTP ON,LOAD,HZ,PLRT,NOR* Set the load of CH1 to 50 ohms: *C1:OUTP LOAD,50* Set the load of CH1 to HiZ: *C1:OUTP LOAD,HZ* Set the polarity of CH1 to normal: *C1:OUTP PLRT,NOR*

Note: * "HiZ" refers to High Z.

DESCRIPTION	Set two channels to open or close output at the same time
COMMAND SYNTAX	OUT_BOTHCH <STATE> <STATE>={ON,OFF}
EXAMPLE	Open two channels of output: *OUT_BOTHCH ON*

3.3 Noise superposition Command

DESCRIPTION	Turn on noise superposition and add noise to the channel at the specified signal-to-noise ratio.
COMMAND SYNTAX	<channel>:NOISE_ADD STATE,ON|OFF,RATIO,<value> or <channel>:NOISE_ADD STATE,ON|OFF,RATIO_DB,<value> <channel>:= {C1, C2}. See the data sheet for the SNR values.
QUERY SYNTAX	<channel>:NOISE_ADD?
RESPONSE FORMAT	<channel>:NOISE_ADD STATE,ON|OFF,RATIO,<value> or <channel>:NOISE_ADD STATE,ON|OFF,RATIO_DB,<value>
EXAMPLE	Turn on the channel-noise superposition and set the SNR to 120.: *C1:NOISE_ADD STATE,ON,RATIO,120*

3.4 Basic Wave Command

DESCRIPTION	This command sets or gets the basic wave parameters.
COMMAND SYNTAX	<channel>:BaSic_WaVe <parameter>,<value> <channel>:={C1, C2}. <parameter>:= {a parameter from the table below}. <value>:={value of the corresponding parameter}.

Parameters	Value	Description
WVTP	<type>	:= {SINE, SQUARE, RAMP, PULSE, NOISE, ARB, DC, PRBS, MULTPULSE ,SEQUENCE,IQ}. If the command doesn't set basic waveform type, WVPT will be set to the current waveform.
FRQ	<frequency>	:= frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC.
PERI	<period>	:= period. The unit is seconds "s". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC.
AMP	<amplitude>	:= amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC.
AMPVRMS	<amplitude>	:= amplitude. The unit is Volts, root-mean-square "Vrms".
AMPDBM	<amplitude>	:= amplitude. The unit is dBm.
OFST	<offset>	:= offset. The unit is volts "V". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE.
SYM	<symmetry>	:={0 to 100}. Symmetry of RAMP. The unit is "%". Only settable when WVTP is RAMP.
DUTY	<duty>	:= {0 to 100}. Duty cycle. The unit is "%". Value depends on frequency. Only settable when WVTP is SQUARE or PULSE.
PHSE	<phase>	:= {0 to 360}. The unit is "degree". Not valid when WVTP is NOISE, PULSE or DC.
STDEV	<stdev>	:= standard deviation of NOISE. The unit is volts "V". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE.

MEAN	<mean>	:= mean of NOISE. The unit is volts "V". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE.
WIDTH	<width>	:= positive pulse width. The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE.
RISE	<rise>	:= rise time (10%~90%). The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE.
FALL	<fall>	:= fall time (90%~10%). The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE.
DLY	<delay>	:= waveform delay. The unit is seconds "s". Refer to the datasheet for the range of valid values.
HLEV	<high level>	:= high level. The unit is volts "V". Not valid when WVTP is NOISE or DC.
LLEV	<low level>	:= low level. The unit is volts "V". Not valid when WVTP is NOISE or DC.
BANDSTATE	<bandwidth switch >	:= {ON,OFF}. Only settable when WVTP is NOISE.
BANDWIDTH	<bandwidth value>	:= noise bandwidth. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE.
LENGTH	<prbs length>	:={3~32}. Actual PRBS length = $2^{LENGTH}-1$. Only settable when WVTP is PRBS.
EDGE	<prbs rise/fall>	:= rise/fall time of PRBS. The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PRBS.
DIFFSTATE	<prbs differential switch>	:={ON, OFF}. State of PRBS differential mode. Only settable when WVTP is PRBS.
BITRATE	<prbs bit rate>	:= PRBS bit rate. The unit is bits-per-second "bps". Refer to the datasheet for the range of valid values. Only settable when WVTP is PRBS.
LOGICLEVEL	<prbs logiclevel rate>	:={ TTL_CMOS, LVTTL_LVCMOS, ECL, LVPECL, LVDS CUSTOM }. Only settable when WVTP is PRBS.
MAX_OUTPUT_AMP	<amplitude>	:=Channel maximum amplitude limit. Refer to the datasheet for the range of valid values.

QUERY SYNTAX	<channel>: BaSic_WaVe? <channel>:= {C1, C2}.
RESPONSE FORMAT	<channel>:BSWV <parameter> <parameter>:= {All the parameters of the current basic waveform}.
EXAMPLE	Change the waveform type of C1 to Ramp: *C1:BSWV WVTP,RAMP* Change the frequency of C1 to 2000 Hz: *C1:BSWV FRQ,2000* Set the amplitude of C1 to 3 Vpp: *C1:BSWV AMP,3* Return parameters of C1 from the device: *C1:BSWV?* Return: *C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V,* *OFST,0V,HLEV,1V,LLEV,-1V,PHSE,0* Set noise bandwidth of C1 to 100 MHz: *C1:BSWV BANDWIDTH,100E6* or *C1:BSWV BANDWIDTH,100000000* Set output amplitude of C1 to 3dBm: *C1:BSWV AMPDBM,3* Set the logic level of C1 to TTL_CMOS: *C1:BSWV LOGICLEVEL,TTL_CMOS*

3.5 PRBS polynomial Command

DESCRIPTION	This command is used to set or obtain PRBS user-defined polynomial parameters. This command is valid only when the basic waveform is PRBS.
COMMAND SYNTAX	< channel >:PRBS:< parameter > < value > < channel >:= {C1, C2}. < value >:= { The values of relevant parameters }

parameter	value	Description
STAT	<STATE>	:={RUN,STOP} ,Turn PRBS playback status on or off.
TYPE	<type>	:={FORMULA,LENGTH},Set the polynomial type as default or custom.
ITEM:Add	<value>	:= {1，2，...，32},Set PRBS add items.
ITEM:Delet	<value>	:= {1，2，...，32},Set PRBS deletion items.
ITEM:Clear	None	:= Clear PRBS polynomials.
SEED	<value>	:= {1，2，...，M},Set the PRBS seed value.

QUERY SYNTAX	< channel >:PRBS:<parameter>? < channel >:={C1, C2}.
EXAMPLES	Enable PRBS playback status of CH1: *C1:PRBS:STAT RUN* Set the PRBS polynomial type of CH1 to custom: *C1:PRBS:TYPE FORMULA* Gets the PRBS polynomial type of CH1: *C1:PRBS:TYPE?* Return: *FORMULA*

3.6 DC amplifier Command

DESCRIPTION	This command sets or gets the status of the amplifier.
COMMAND SYNTAX	< channel >:DcUFAmplifier <state> < channel >:= {C1, C2}. < state >:= { ON,OFF }
QUERY SYNTAX	< channel >:DcUFAmplifier?
EXAMPLES	Turn on the amplifier of CH1: *C1:DcUFAmplifier ON* Inquire about the amplifier status of CH2: *C2:DcUFAmplifier?* Return: *C2:DcUFAmplifier,OFF*

3.7 Harmonic Command

DESCRIPTION	This command sets or gets the harmonic parameters. Only available when the basic wave is SINE.
COMMAND SYNTAX	< channel >:HARMonic < parameter >,< value > < channel >:= {C1, C2}. < value >:= { The values of relevant parameters }

parameter	value	Description
HARMSTATE	<STATE>	:={ON, OFF} Turn harmonics on or off
HARMTYPE	<TYPE>	: {EVEN , ODD , ALL} harmonic types.
HARMORDER	<NUM>	: {1, 2,..., M}, where M is the maximum supported series.
HARMPHASE	<PHASE>	:= {0~360}, the unit is "degree"
HARMAMP	<VALUE>	:= amplitude of specified harmonic. the unit is "dBc".
HARMDBC	<VALUE>	:= amplitude of specified harmonic. the unit is "dBc".

QUERY SYNTAX	< channel >:HARMonic? < channel >:={C1, C2}.
EXAMPLES	Enable the harmonic function of CH1: *C1:HARM HARMSTATE,ON* Set the 2nd harmonic of CH1 to -6 dBc: *C1:HARM HARMORDER,2,HARMDBC,-6* Get the harmonic information of CH1: *C1:HARM?* Return: *C1:HARM ,HARMSTATE,ON,HARMTYPE,EVEN,HARMORDER,2,HARMAMP,2.004748935V,HARMDBC,-6dBc,HARMPHASE,0*

3.8 MultPulse Command

DESCRIPTION	This command sets or gets the multpulse parameters. Only available when the basic wave is multpulse.
COMMAND SYNTAX	Format: < channel >:MultiPulse:<parameter> < value > < channel >:={C1, C2}. < value >:= {The values of relevant parameters1 }

Parameter1	value	Description
Count	<count>	:= Pulse count. Refer to the datasheet for the range of valid values.
Source	<source>	:={ INT, EXT, MAN, TIMER}, Trigger source. INT means internal trigger source, EXT means external trigger source, MAN means manual trigger, and TIMER means timer trigger.
TIMer	<timer>	Set the timing time, which is effective when the trigger source is a timer.
Dlay	<delay>	Set the trigger delay, which is effective when the trigger source is manual or external.
Edge	<edge>	:={UP, DOWN}, Set the external trigger edge, which is effective when the trigger source is external.
Item#:PosWidth	<width>	Set the positive pulse width of the pulse, and # indicates the segment number of the pulse.
Item#:NegWidth	<width>	Set the negative pulse width of the pulse, and # indicates the segment number of the pulse.
Trigger	None	Set manual trigger, which takes effect when the trigger source is manual.

QUERY SYNTAX	Format: < channel >:MultiPulse:< parameter >?
EXAMPLES	Set the count of CH1 to 3: *C1:MultiPulse:Count 3* Get the count of CH1: *C1:MultiPulse:Count?* Return: *3*

	Set the positive pulse width of channel 1 pulse 1 to 20μs: *C1:MultiPulse:Item1:PosWidth 0.00002* Get the positive pulse width of channel 1 pulse 1: *C1:MultiPulse:Item1:PosWidth?* Return: *2e-05*

3.9 Arbitrary Wave Command

3.9.1 Arbitrary Wave Switch Command

DESCRIPTION	This command sets and gets the arbitrary waveform type. Note: The index number in the command syntax and the response format of the query omits the character and directly uses the value to represent the index number.
COMMAND SYNTAX	Format1: <channel>:ArbWaVe INDEX,<index> Format2: <channel>:ArbWaVe NAME,<name> Format3: <channel>:ArbWaVe NAME,<path> <channel>:= {C1, C2}. <index>: the index of the arbitrary waveform from the table below. <name>: the name of the arbitrary waveform from the table below. <path>: the path of waveform
QUERY SYNTAX	<channel>:ARbWaVe? <channel>:= {C1, C2}.
RESPONSE FORMAT	<channel>:ARWV INDEX,<index>,NAME,<name>
EXAMPLE	Set CH1 current waveform by index 2: *C1:ARWV INDEX,2* Read CH1 current waveform: *C1:ARWV?* Return: *C1:ARWV INDEX,2,NAME,StairUp* Set CH1 current waveform to sine by name. *C1:ARWV NAME,sine* Set the waveform of ch1 through the waveform path: *C1:ARWV NAME,"wave1.bin"* *C1:ARWV NAME,"wave3.csv"* *C1:ARWV NAME,"U-disk0/wave1.bin"*
NOTE	The specific path refers to the path in the file manager
RELATED COMMANDS	STL

Index	Name	Index	Name	Index	Name	Index	Name
0	Sine	51	AttALT	102	LFPulse	153	Duty18
1	Noise	52	RoundHalf	103	Tens1	154	Duty20
2	StairUp	53	RoundsPM	104	Tens2	155	Duty22
3	StairDn	54	BlaseiWave	105	Tens3	156	Duty24
4	Stairud	55	DampedOsc	106	Airy	157	Duty26
5	Ppulse	56	SwingOsc	107	Besselj	158	Duty28
6	Npulse	57	Discharge	108	Bessely	159	Duty30
7	Trapezia	58	Pahcur	109	Dirichlet	160	Duty32
8	Upramp	59	Combin	110	Erf	161	Duty34
9	Dnramp	60	SCR	111	Erfc	162	Duty36
10	ExpFal	61	Butterworth	112	Erfclnv	163	Duty38
11	ExpRise	62	Chebyshev1	113	Erflnv	164	Duty40
12	Logfall	63	Chebyshev2	114	Laguerre	165	Duty42
13	Logrise	64	TV	115	Legend	166	Duty44
14	Sqrt	65	Voice	116	Versiera	167	Duty46
15	Root3	66	Surge	117	Weibull	168	Duty48
16	X^2	67	Radar	118	LogNormal	169	Duty50
17	X^3	68	Ripple	119	Laplace	170	Duty52
18	Sinc	69	Gamma	120	Maxwell	171	Duty54
19	Gaussian	70	StepResp	121	Rayleigh	172	Duty56
20	Dlorentz	71	BandLimited	122	Cauchy	173	Duty58
21	Haversine	72	CPulse	123	CosH	174	Duty60
22	Lorentz	73	CWPulse	124	CosInt	175	Duty62
23	Gauspuls	74	GateVibr	125	CotH	176	Duty64
24	Gmonopuls	75	LFMPulse	126	CscH	177	Duty66
25	Tripuls	76	MCNoise	127	SecH	178	Duty68
26	Cardiac	77	AM	128	SinH	179	Duty70
27	Quake	78	FM	129	SinInt	180	Duty72
28	Chirp	79	PFM	130	TanH	181	Duty74
29	Twotone	80	PM	131	ACosH	182	Duty76
30	SNR	81	PWM	132	ASecH	183	Duty78
31	Hamming	82	EOG	133	ASinH	184	Duty80
32	Hanning	83	EEG	134	ATanH	185	Duty82
33	Kaiser	84	EMG	135	ACsch	186	Duty84
34	Blackman	85	Pulseilogram	136	ACoth	187	Duty86

35	Gausswin	86	ResSpeed	137	Bartlett	188	Duty88
36	Triang	87	ECG1	138	BohmanWin	189	Duty90
37	BlackmanH	88	ECG2	139	ChebWin	190	Duty92
38	Bartlett	89	ECG3	140	FlattopWin	191	Duty94
39	Tan	90	ECG4	141	ParzenWin	192	Duty96
40	Cot	91	ECG5	142	TaylorWin	193	Duty98
41	Sec	92	ECG6	143	TukeyWin	194	Duty99
42	Csc	93	ECG7	144	Duty01	195	demo1_375
43	Asin	94	ECG8	145	Duty02	196	demo1_16k
44	Acos	95	ECG9	146	Duty04	197	demo2_3k
45	Atan	96	ECG10	147	Duty06	198	demo2_16k
46	Acot	97	ECG11	148	Duty08		
47	Square	98	ECG12	149	Duty10		
48	SineTra	99	ECG13	150	Duty12		
49	SineVer	100	ECG14	151	Duty14		
50	AmpALT	101	ECG15	152	Duty16		

3.9.2 Arb Data Command

DESCRIPTION	This command sets the arbitrary waveform data.
COMMAND SYNTAX	<channel>:WVDT,<parameter>,<value> <channel>:= {C1, C2}. <parameter>:= {a parameter from the table below}. <value>:= {value of the corresponding parameter}.

Parameters	Value	Description
WVNM	<wave_name>	:= waveform name.
LENGTH	<length>	:= the number of waveform bytes, the valid range depends on the model. See note 1 below for the details. This parameter is not necessary for the "X" series
FREQ	<frequency>	:= frequency. The unit is Hertz "Hz".
AMPL	<amplifier>	:= amplitude. The unit is volts, peak-to-peak "Vpp".
OFST	<offset>	:= offset. The unit is volts "V".
PHASE	<phase>	:= phase. The unit is "degree".
WAVEDATA	<wave data>	:= waveform data. Data written to waveform file

QUERY SYNTAX	Format 1: WVDT? Mn Format 2: WVDT? USER,<wave_name> <wave name>:={The name of user-defined waveform}. Format3: WVDT? USER,<PATH>,< wave_name> < Path > specify storage path, such as USB flash disk path. See the following example for network storage path < waveform name >: = {user defined waveform name}
EXAMPLE	*C1:WVDT WVNM,wave1,WAVEDATA, b'0x6000c0006000'.*

Notes1:

(1) The path query function must be under the Local or USB flash disk directory. When using the command, English quotation marks must be added at both ends of the path.

(2) The top-level directory of USB flash disk can be obtained in the file manager.

(3) Do not specify a path. The default is the local path (format 2).

(4) Try to fix the directory where waveform files are stored to reduce the waiting time caused by file retrieval.

3.10 Modulate Wave Command

DESCRIPTION	This command sets or gets the modulation parameters.
COMMAND SYNTAX	<channel>:MoDulateWaVe <type> <channel>:MoDulateWaVe <parameter>,<value> <channel>:={C1, C2} <type>:= {AM,DSBAM,FM,PM,PWM,ASK,FSK,PSK}. <parameter>:= {a parameter from the table below}. <value>:= {value of the from the table below}.

Parameters	Value	Description
RSTAT	<state>	:= {RUN, STOP}, Set the MOD playback status. Please refer to the example for specific usage.
STATE	<state>	:={ON, OFF}. Enable or disable modulation. STATE must be set to ON before you set or read other parameters of the modulation.
AM,SRC	<src>	:= {INT, EXT, CH1, CH2}. AM modulation source. Where' CH1' takes effect only when setting the CH2 modulation source, and' CH2' takes effect only when setting the CH1 modulation source.
AM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. AM modulation wave. Only settable when SRC is INT.
AM,FRQ	<AM frequency>	:= AM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
AM,DEPTH	<depth>	:= {0 to 120}. AM depth. The unit is "%". Only settable when SRC is INT.
DSBAM,SRC	<src>	:= {INT, EXT, CH1, CH2}. DSB-AM modulation source.
DSBAM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. DSB AM modulation wave. Only settable when SRC is INT.
DSBAM,FRQ	<DSB-AM frequency>	:= DSB-AM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.

FM,SRC	<src>	:= {INT, EXT, CH1, CH2}. FM modulation source. Where' CH1' takes effect only when setting the CH2 modulation source, and' CH2' takes effect only when setting the CH1 modulation source.
FM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. FM modulation wave. Only settable when SRC is INT.
FM,FRQ	<FM frequency>	:= FM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
FM,DEVI	<FM frequency deviation >	:= {0 to carrier frequency}. FM frequency deviation. The value depends on the difference between the carrier frequency and the bandwidth frequency. Only settable when SRC is INT.
PM,SRC,	<src>	:= {INT, EXT, CH1, CH2}. PM modulation source. Where' CH1' takes effect only when setting the CH2 modulation source, and' CH2' takes effect only when setting the CH1 modulation source.
PM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. PM modulation wave. Only settable when SRC is INT.
PM,FRQ	<PM frequency>	:= PM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
PM,DEVI	<PM phase offset>	:= {0 to 360}. PM phase deviation. The unit is "degree". Only settable when SRC is INT.
PWM,SRC	<src>	:= {INT, EXT, CH1, CH2}. PWM modulation source. Where' CH1' takes effect only when setting the CH2 modulation source, and' CH2' takes effect only when setting the CH1 modulation source.
PWM,FRQ	<PWM frequency>	:= PWM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
PWM,DEVI	<PWM dev>	:= Duty cycle deviation. The unit is "%". Value depends on the carrier duty cycle.
PWM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP,

		NOISE, ARB}. PWM modulation wave. Only settable when SRC is INT.
ASK,SRC	<src>	:= {INT, EXT}. ASK modulation source.
ASK,KFRQ	< key frequency>	:= ASK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
FSK,SRC	<src>	:= {INT, EXT}. FSK modulation source.
FSK,KFRQ	< key frequency>	:= FSK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
FSK,HFRQ	<FSK_hop_freq>	:= FSK hop frequency. The same with basic wave frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
PSK,SRC	<src>	:= {INT, EXT}. PSK modulation source.
PSK,KFRQ	< key frequency>	:= PSK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.
PSK,PLRT	<polarity>	:={POS,NEG}.
MDSP,ARB,INDEX	<index>	:= {0 to 198}. Set the modulation waveform as a built-in waveform, which can be set when the modulation waveform is Arb.
MDSP,ARB,NAME	<name>	Name of the waveform, set the modulation waveform as the saved waveform, which can be set when the modulation waveform is Arb.
CARR,WVTP	<wave type>	:= {SINE, SQUARE, RAMP, ARB, PULSE}. Carrier waveform type.
CARR,FRQ	<frequency>	:= carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
CARR,PHSE	<phase>	:= {0 to 360}. Carrier phase. The unit is "degree".
CARR,AMP	<amplitude>	:= carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values.
CARR,OFST	<offset>	:= carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values.
CARR,SYM	<symmetry>	:= {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is "%".
CARR,DUTY	<duty>	:= {0 to 100}. Carrier duty cycle when the carrier is SQUARE or PULSE. The unit is "%".

CARR,RISE	<rise>	:= rise time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values.
CARR,FALL	<fall>	:= fall time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values.
CARR,DLY	<delay>	:= pulse delay when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values.

QUERY SYNTAX	<channel>:MoDulateWaVe? <channel>:={C1, C2}.
RESPONSE FORMAT	<channel>:MDWV <parameter> <parameter>:= {all parameters of the current modulation}.
EXAMPLE	Set CH1 modulation state to on: *C1:MDWV STATE,ON* Set the CH1 playback status to Run.: *C1:MDWV:RSTAT RUN* Set CH1 modulation type to AM: *C1:MDWV AM* Set modulation to AM, and the modulating wave type to sine wave: *C1:MDWV AM,MDSP,SINE* Read CH1 modulation parameters when STATE is ON: *C1:MDWV?* Return: *C1:MDWV AM,STATE,ON,MDSP,SINE,SRC,INT,FRQ,100HZ,DEPTH,100,CARR,WVTP,RAMP,FRQ,1000HZ,AMP,4V,AMPVRMS,1.15473Vrms,OFST,0V,PHSE,0,SYM,50* Read CH1 modulate wave parameters when STATE is OFF: *C1:MDWV?* Return: *C1:MDWV STATE,OFF* Set CH1 FM frequency to 1000 Hz: *C1:MDWV FM,FRQ,1000* Set CH1 carrier to SINE: *C1:MDWV CARR,WVTP,SINE* Set CH1 carrier frequency to 1000 Hz: *C1:MDWV CARR,FRQ,1000*

3.11 Sweep Wave Command

3.11.1 <channel>: SweepWaVe <para>,<value>

DESCRIPTION	This command sets or gets the sweep parameters.
COMMAND SYNTAX	<channel>:SweepWaVe <parameter>,<value> <channel>:={C1, C2} <parameter>:= {a parameter from the table below} <value>:={value of the corresponding parameter}

Parameters	Value	Description
RSTAT	<state>	:= {RUN, STOP}, Set the Sweep playback status. Please refer to the example for specific usage.
STATE	<state>	:={ON, OFF}. Enable or disable sweep. STATE must be set to ON before you set or read other parameters of the sweep.
TIME	<time>	:= sweep time. The unit is seconds "s". Refer to the datasheet for the range of valid values.
START	<start_freq>	:= start frequency. The same with basic wave frequency. The unit is Hertz "Hz".
STOP	<stop_freq>	:= stop frequency. The same with basic wave frequency. The unit is Hertz "Hz".
CENTER	<center_freq>	:= center frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
SPAN	<span_freq>	:= frequency span. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
SWMD	<sweep_mode>	:= {LINE,LOG,STEP}, in which LINE refers to Linear ,LOG refers to Logarithmic and STEP refers to step.
STEPNUM	<value>	:={2 to 1024}. The number of steps, which is effective when the scanning type is step.
DIR	<direction>	:= {UP, DOWN, UP_DOWN}. Sweep direction.
SYM	<symmetry>	:={0% to 100%}. The symmetry when sweep direction is up_down.
TRSR	<trig_src>	:= {EXT, INT, MAN}. Trigger source. EXT refers to External, INT refers to Internal and MAN refers to Manual.
MTRIG		:= send a manual trigger. Only valid when TRSR is MAN.

TRMD	<trig_mode>	:= {ON, OFF}. State of trigger output. If TRSR is EXT, the parameter is invalid.
EDGE	<edge>	:={RISE, FALL}. Available trigger edge. Only valid when TRSR is EXT or MAN.
MARK_STATE	<state>	:= {ON, OFF}. Set the frequency marker status on or off.
MARK_FREQ	<value>	Marking frequency, effective when the scanning type is linear or logarithmic, and the marking range is from the starting frequency to the ending frequency.
MARK_STEP	<value>	:={2 to 1024}. Mark step, which is effective when the scanning type is step.
IDLE_FREQ	<idle freq>	:= {ZERO_FREQ,START_FREQ,STOP_FREQ} , Idle frequency. ZERO_FREQ means zero level, START_FREQ means starting frequency, and STOP_FREQ means ending frequency. Effective only when TRSR is set to EXT or MAN.
OUTPUT_MODE	<output mode>	:= { SINGLE, CONTINU},Out type. SINGLE indicates that the output mode is single sweep, and CONTINU indicates that the output mode is continuous sweep. Effective only when TRSR is set to EXT or MAN.
CURFREQ	None	This parameter is used to query the frequency of current output. Please refer to the example for specific usage.
CARR, WVTP	<wave type>	:= {SINE, SQUARE, RAMP, ARB}. Carrier waveform type. Modulation is not available if the carrier is PULSE, NOISE, or DC.
CARR, FRQ	<frequency>	:= carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
CARR, PHSE	<phase>	:= {0 to 360}. Carrier phase. The unit is "degree".
CARR, AMP	<amplitude>	:= carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values.
CARR, OFST	<offset>	:= carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values.
CARR, SYM	<symmetry>	:= {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is percent "%".
CARR, DUTY	<duty>	:= {0 to 100}. Carrier duty cycle when the carrier is SQUARE. The unit is "%".

QUERY SYNTAX <channel>:SWeepWaVe?

<channel>:= {C1, C2}.

RESPONSE FORMAT	<channel>:SWWV <parameter> <parameter>:= {All parameters of the current sweep wave}
EXAMPLE	Set CH1 sweep state to ON: *C1:SWWV STATE,ON* Set the CH1 playback status to Run: *C1:SWEep:RSTAT RUN* Set CH1 sweep time to 1 s: *C1:SWWV TIME,1* Set CH1 stop frequency to 1000 Hz: *C1:SWWV STOP,1000* Set the trigger source of CH1 to Manual: *C1:SWWV TRSR,MAN* Send a manual trigger to CH1: *C1:SWWV MTRIG* Read CH2 sweep parameters when STATE is ON: *C2:SWWV?* Return: *C2:SWWVSTATE,ON,TIME,1S,STOP,1500HZ,START,500HZ,CENTER,1000HZ,SPAN,1000HZ,TRSR,INT,TRMD,OFF,SWMD,LINE,DIR,UP,SYM,0,MARK_STATE,OFF,MARK_FREQ,1000HZ,CARR,WVTP,SINE,FRQ,1000HZ,AMP,4V,AMPVRMS,1.41421Vrms,OFST,0V,PHSE,0* Read CH2 sweep parameters when STATE is OFF: *C2:SWWV?* Return: *C2:SWWV STATE,OFF* Set CH1 the FreqMarker of sweep to 1kHz *C1:SWWV MARK_STATE,ON,MARK_FREQ,1000* Read the output frequency of CH1: *C1:SWEep:CURFREQ?* Return: *505.05253904968*

3.11.2 <channel>:SWEep:TYPE <type>

DESCRIPTION	This command sets or gets the sweep type.
COMMAND SYNTAX	<channel>:SWEep:TYPE <type> <channel>:= {C1, C2}. <type>:= {FREQ, AMP,}.
QUERY SYNTAX	<channel>: SWEep:TYPE? <channel>:= {C1, C2}.
EXAMPLE	Set CH1 sweep type to frequency sweep. *:C1:SWEep:TYPE FREQ* Get CH1 sweep type. *:C1:SWEep:TYPE?* Return: *FREQ*

3.11.3 <channel>:SWEep:SAMPlitude <value>

DESCRIPTION	This command sets or gets the sweep start amplitude.
COMMAND SYNTAX	<channel>:SWEep:SAMPlitude <value> <channel>:= {C1, C2}. <value>:= floating bumber with volt unit
QUERY SYNTAX	<channel>: SWEep:SAMPlitude? <channel>:= {C1, C2}
EXAMPLE	Sets CH1 sweep start amplitude to 100mv *:C1:SWEep:SAMPlitude 0.1* Gets CH1 sweep start amplitude. *:C1:SWEep:SAMPlitude?* Return: *"0.1"*

3.11.4 <channel>:SWEep:EAMPlitude <value>

DESCRIPTION	This command sets or gets the sweep stop amplitude.
COMMAND SYNTAX	<channel>:SWEep:EAMPlitude <value> <channel>:= {C1, C2}. <value>:= floating bumber with volt unit
QUERY SYNTAX	<channel>: SWEep:EAMPlitude? <channel>:= {C1, C2}..

EXAMPLE	Sets CH1 sweep stop amplitude to 900mv *:C1:SWEep:EAMPlitude 0.9* Gets CH1 sweep stop amplitude. *:C1:SWEep:EAMPlitude?* Return: *"0.9"*

3.11.5 <channel>:SWEep:CAMPlitude <value>

DESCRIPTION	This command sets or gets the sweep center amplitude.
COMMAND SYNTAX	<channel>:SWEep:CAMPlitude <value> <channel>:= {C1, C2}. <value>:= floating bumber with volt unit
QUERY SYNTAX	<channel>: SWEep:CAMPlitude? <channel>:= {C1, C2}..
EXAMPLE	Sets CH1 sweep center amplitude to 500mv *:C1:SWEep:CAMPlitude 0.5* Gets CH1 sweep center amplitude. *:C1:SWEep:CAMPlitude?* Return: *"0.5"*

3.11.6 <channel>:SWEep:ASPan <value>

DESCRIPTION	This command sets or gets the sweep amplitude span.
COMMAND SYNTAX	<channel>:SWEep:ASPan <value> <channel>:= {C1, C2}. <value>:= floating bumber with volt unit
QUERY SYNTAX	<channel>: SWEep:ASPan? <channel>:= {C1, C2}..
EXAMPLE	Sets CH1 sweep amplitude span to 800mv *:C1:SWEep:ASPan 0.8* Gets CH1 sweep amplitude span. *:C1:SWEep:ASPan?* Return: *"0.8"*

3.12 Burst Wave Command

DESCRIPTION	This command sets or gets the burst wave parameters.
COMMAND SYNTAX	<channel>:BursTWaVe <parameter>,<value> <channel>:= {C1, C2}. <parameter>:= {a parameter from the table below}. <value>:= {value of the corresponding parameter}.

Parameters	Value	Description
RSTAT	<state>	:= {RUN, STOP}, Set the Burst playback status. Please refer to the example for specific usage.
STATE	<state>	:= {ON, OFF}. Enable or disable burst. STATE must be set to ON before you set or read other parameters of the burst.
PRD	<period>	:= burst period. Refer to the datasheet for the range of valid values. The unit is seconds “s”
STPS	<start_phase>	:= {0 to 360}. Start phase of the carrier. The unit is "degree". Not valid when the carrier is NOISE or PULSE.
GATE_NCYC	<burst_mode>	:= {GATE, NCYC}. Burst mode. Not valid when the carrier is NOISE.
TRSR	<trig_src>	:= {EXT, INT, MAN}. Trigger source. EXT refers to External, INT refers to Internal and MAN refers to Manual.
MTRIG		:= send a manual trigger. Only when TRSR is MAN, the parameter is valid.
DLAY	<delay>	:= trigger delay. The unit is seconds "s". Refer to the datasheet for the range of valid values. Available when GATE_NCYC is NCYC. Not valid when the carrier is NOISE.
PLRT	<polarity>	:= {NEG, POS}. Gate polarity. Negative or Positive.
TRMD	<trig_mode>	:= {RISE, FALL, OFF}. Trigger out mode. Available when GATE_NCYC is NCYC and TRSR is INT or MAN. Not valid when the carrier is NOISE.
EDGE	<edge>	:= { RISE, FALL}. Available trigger edge. Available when GATE_NCYC is NCYC and TRSR is EXT. Not valid when the carrier is NOISE.
TIME	<circle_ time>	:={INF, 1, 2,..., M}, where M is the maximum supported Ncycle number which depends on the model; INF sets the burst to Infinite mode. Available when GATE_NCYC is NCYC. Not valid when the carrier is NOISE.

HVALue	<hold value>	:= {ZERo,SVALue,EVALue}, Idle level. ZERo represents the intermediate value, SVALue represents the starting value and EVALue represents the ending value. Please refer to the example for specific usage.
COUNTER	<counter>	:=Burst count, Only valid when TRSR is EXT or MAN.
CARR, WVTP	<wave type>	:= {SINE, SQUARE, RAMP, ARB, PULSE, NOISE}. Carrier waveform type.
CARR, FRQ	<frequency>	:= carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values.
CARR, PHSE	<phase>	:= {0 to 360}. Carrier phase. The unit is "degree".
CARR, AMP	<amplitude>	:= carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values.
CARR, OFST	<offset>	:= carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values.
CARR, SYM	<symmetry>	:= {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is "%".
CARR, DUTY	<duty>	:= {0 to 100}. Carrier duty cycle when the carrier is SQUARE or PULSE. The unit is "%".
CARR, RISE	<rise>	:= rise time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values.
CARR, FALL	<fall>	:= fall time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values.
CARR, DLY	<delay>	:= pulse delay when the carrier is PULSE. The unit is seconds “s”. Refer to the datasheet for the range of valid values.
CARR,STDEV	<stdev>	:= standard deviation of NOISE. The unit is volts “V”. Refer to the datasheet for the range of valid values.
CARR, MEAN	<mean>	:= mean of NOISE. The unit is volts “V”. Refer to the datasheet for the range of valid values.

QUERY SYNTAX	<channel>:BTWV(BursTWaVe)? <channel>:={C1, C2}
RESPONSE FORMAT	<channel>:BTWV <parameter> <parameter>:={All parameters of the current burst wave.}
EXAMPLE	Set CH1 burst state to ON *C1:BTWV STATE,ON* Set the CH1 playback status to Run: *C1:BURSt:RSTAT RUN* Set CH1 burst period to 1 s.

C1:BTWV PRD,1

Set CH1 burst delay to 1 s

C1:BTWV DLAY,1

Set CH1 burst to infinite

C1:BTWV TIME,INF

Set the idle level of CH1 to the end value

C1:BURSt:HVALue EVALue

Read CH2 burst parameters when the STATE is ON.

C2:BTWV?

Return:

C2:BTWV STATE,ON,PRD,0.01S,STPS,0,TRSR,INT,
TRMD,OFF,TIME,1,DLAY,2.4e-07S,GATE_NCYC,NCYC,
CARR,WVTP,SINE,FRQ,1000HZ,AMP,4V,OFST,0V,PHSE,0

Read CH2 burst parameters when the STATE is OFF.

C2:BTWV?

Return:

C2:BTWV STATE,OFF

3.13 Sequence Command

3.13.1 <channel>:SEQuence <switch>

DESCRIPTION	This command is used to set (query) the output status of the sequence
COMMAND SYNTAX	[:SOURce]:<channel>:SEQuence <switch> <channel>:={C1,C2}. <switch>:={ON,OFF}.
QUERY SYNTAX	[:SOURce]:<channel>:SEQuence? <channel>:={C1,C2}.
EXAMPLE	Turn on the sequence output of channel 1 *:C1:SEQuence ON* Read the sequence output status of channel 1 *:C1:SEQuence?* Return: *ON*

3.13.2 <channel>:SEQuence:SRATe

DESCRIPTION	This command is used to set (query) the sampling rate of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SRATe <value> <channel>:={C1,C2}. <value>:=sampling rate, Unit sa/s
QUERY SYNTAX	<channel>:SEQuence:SRATe? <channel>:={C1,C2}.
EXAMPLE	Set the sampling rate of channel 1 sequence to 16385000 *:C1:SEQuence:SRATe 16385000* Query the sampling rate of channel 1 sequence *:C1:SEQuence:SRATe?* Return: *16385000*

3.13.3 <channel>:SEQuence:STATe

DESCRIPTION	This command is used to set (query) the running status of the sequence
COMMAND SYNTAX	<channel>:SEQuence:STATe <state> <channel>:={C1,C2}. <state>:={RUN,STOP}
QUERY SYNTAX	<channel>:SEQuence:STATe? <channel>:={C1,C2}.
EXAMPLE	Set the status of channel 1 sequence to "run" *:C1:SEQuence:STATe RUN* Query the status of channel 1 sequence *:C1:SEQuence:STATe?* Return: *RUN*

3.13.4 <channel>:SEQuence:SCALe

DESCRIPTION	This command is used to set (query) the output amplitude of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SCALe <scale> <channel>:={C1,C2}. <scale>:={1 to 100}
QUERY SYNTAX	<channel>:SEQuence:SCALe? <channel>:={C1,C2}.
EXAMPLE	Set the output amplitude of channel 1 sequence to 80% *:C1:SEQuence:SCALe 80* Query the output amplitude of channel 1 sequence *:C1:SEQuence:SCALe?* Return: *"80.000000%"*

3.13.5 <channel>:SEQuence:OFFset

DESCRIPTION	This command is used to set (query) the offset of the sequence
COMMAND SYNTAX	<channel>:SEQuence:OFFset <value> <channel>:={C1,C2}.

	<value>:={-8 to 8}
QUERY SYNTAX	<channel>:SEQuence:OFFset? <channel>:={C1,C2}.
EXAMPLE	Set the offset of channel 1 sequence to 1 *:C1:SEQuence:OFFset 1* Query the output offset of channel 1 sequence *:C1:SEQuence:OFFset?* Return: *1*

3.13.6 <channel>:SEQuence:SAVe

DESCRIPTION	This command is used to save the sequence waveform to a file
COMMAND SYNTAX	<channel>:SEQuence:SAVe<path> <channel>:={C1,C2}. <path>:= Complete path of waveform
EXAMPLE	Save the sequence waveform to “Local/sequence.awgx” *:C1:SEQuence:SAVe "sequence.awgx"*

3.13.7 <channel>:SEQuence:RECall

DESCRIPTION	This command is used to load sequence waveforms from a file
COMMAND SYNTAX	<channel>:SEQuence:RECall <path> <channel>:={C1,C2}. <path>:= Complete path of waveform
EXAMPLE	From “Local/sequence.awgx” loading sequence waveform *:C1:SEQuence:RECall "sequence.awgx"*

3.13.8 <channel>:SEQuence:RMODe <mode>

DESCRIPTION	This command is used to set (query) the running mode of the sequence
COMMAND SYNTAX	<channel>:SEQuence:RMODe <mode> <channel>:={C1,C2}. <mode>:={CONT, STEP }.

QUERY SYNTAX	<channel>:SEQuence:RMODe? <channel>:={C1,C2}.
EXAMPLE	Set the operation mode of channel 1 sequence to continuous operation *:C1:SEQuence:RMODe CONT* Query the operation mode of channel 1 sequence *:C1:SEQuence:RMODe?* Return: *CONT*

3.13.9 <channel>:SEQuence:STARTNumb

DESCRIPTION	This command is used to set (query) the starting segment of sequence
COMMAND SYNTAX	<channel>:SEQuence:STARTNumb <num> <channel>:={C1,C2}. <num>:= segment number.
QUERY SYNTAX	<channel>:SEQuence:STARTNumb? <channel>:={C1,C2}.
EXAMPLE	Set the starting segment of channel 1 sequence to 3 *:C1:SEQuence:STARTNumb 3* Query the starting segment of channel 1 sequence *:C1:SEQuence:STARTNumb?* Return: *3*

3.13.10 <channel>:SEQuence:INTPo

DESCRIPTION	This command is used to set (query) the interpolation method of the sequence.
COMMAND SYNTAX	<channel>:SEQuence:INTPo <type> <channel>:={C1,C2}. <type>:= { LINE,HOLD,SINC,SINC13,SINC27 }
QUERY SYNTAX	<channel>:SEQuence:INTPo? <channel>:={C1,C2}.
EXAMPLE	Set the interpolation method of channel 1 sequence to LINE *:C1:SEQuence:INTPo LINE*

	Query the interpolation method of channel 1 sequence *:C1:SEQuence:INTPo?* Return: *LINE*

3.13.11 <channel>: SEQuence:TRIGger:SOURce

DESCRIPTION	This command is used to set (query) the trigger mode when the sequence runs
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:SOURce <src> <channel>:={C1,C2}. <src>:={MAN, EXT}.
QUERY SYNTAX	<channel>:SEQuence:TRIGger:SOURce? <channel>:={C1,C2}.
EXAMPLE	Set the trigger mode of channel 1 sequence to external trigger *:C1:SEQuence:TRIGger:SOURce EXT* Query the trigger mode of channel 1 sequence *:C1:SEQuence:TRIGger:SOURce?* Return: *EXT*

3.13.12 <channel>:SEQuence:TRIGger:SLOPe

DESCRIPTION	This command is used to set (query) the trigger edge of the external trigger of the sequence
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:SLOPe <slope> <channel>:={C1,C2}. <slope>:={RISe,FALL}.
QUERY SYNTAX	<channel>:SEQuence:TRIGger:SLOPe? <channel>:={C1,C2}.
EXAMPLE	Set the trigger polarity of the external trigger of channel 1 sequence as the rising edge *:C1:SEQuence:TRIGger:SLOPe RISe* Query trigger polarity of the external trigger of channel 1 sequence *:C1:SEQuence:TRIGger:SLOPe?* Return: *RISe*

3.13.13 <channel>:SEQuence:TRIGger:TRIG

DESCRIPTION	This command immediately triggers a sequence output
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:TRIG <channel>:={C1,C2}
EXAMPLE	Immediately trigger the sequence output of channel 1 *:C1:SEQuence:TRIGger:TRIG*

3.13.14 <channel>:SEQuence:TRIGger:HOLD

DESCRIPTION	This command is used to set (query) the hold valuel of the sequence.
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:HOLD <type> <channel>:={C1,C2}. <type>:={END,MID,START}.
QUERY SYNTAX	<channel>:SEQuence:TRIGger:HOLD? <channel>:={C1,C2}.
EXAMPLE	Set the hold value of channel 1 sequence as the MID *:C1:SEQuence:TRIGger:HOLD MID* Query hold value of channel 1 sequence *:C1:SEQuence:TRIGger:HOLD?* Return: *MID*

3.13.15 <channel>:SEQuence:TRIGger:Delay

DESCRIPTION	This command is used to set (query) the trigger delay of the sequence.
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:Delay <time> <channel>:={C1,C2}. <time>:=trigger delay.
QUERY SYNTAX	<channel>:SEQuence:TRIGger:Delay? <channel>:={C1,C2}.
EXAMPLE	Set the trigger delay of channel 1 sequence as the 1us *:C1:SEQuence:TRIGger:Delay 0.000001* Query trigger delay of channel 1 sequence *:C1:SEQuence:TRIGger:Delay?* Return: *1e-06*

3.13.16 <channel>:SEQuence:TRIGger:Out

DESCRIPTION	This command is used to set (query) the trigger out of the sequence.
COMMAND SYNTAX	<channel>:SEQuence:TRIGger:Out <type> <channel>:={C1,C2}. <type>:={UP,DOWN,OFF}
QUERY SYNTAX	<channel>:SEQuence:TRIGger:Out? <channel>:={C1,C2}.
EXAMPLE	Set the trigger out of channel 1 sequence as the DOWN *:C1:SEQuence:TRIGger:Out DOWN* Query trigger out of channel 1 sequence *:C1:SEQuence:TRIGger:Out?* Return: *DOWN*

3.13.17 <channel>:SEQuence:INCReasing

DESCRIPTION	This command is used to set (query) the interpolation method of the sequence
COMMAND SYNTAX	<channel>:SEQuence:INCReasing <mode> <channel>:={C1,C2}. <mode>:={INT,ZERo,HLAS,DUPL}
QUERY SYNTAX	<channel>:SEQuence:INCReasing? <channel>:={C1,C2}.
EXAMPLE	Set the interpolation mode of channel 1 sequence to linear interpolation *C1:SEQuence:INCReasing INT* Query the interpolation mode of channel 1 sequence *C1:SEQuence:INCReasing?* Return: *INT*

3.13.18 <channel>:SEQuence:DECReasing

DESCRIPTION	This command is used to set (query) the sampling method of the sequence
COMMAND SYNTAX	<channel>:SEQuence:DECReasing <mode>

	<channel>:={C1,C2}. <mode>:={DECi,CTAi,CHEa}
QUERY SYNTAX	<channel>:SEQuence:DECReasing? <channel>:={C1,C2}.
EXAMPLE	Set the sampling method of channel 1 sequence to linear sampling *:C1:SEQuence:DECReasing DECi* Query the sampling method of channel 1 sequence *:C1:SEQuence:DECReasing?* Return: *DECi*

3.13.19 <channel>:SEQuence:SEGMent:Add

DESCRIPTION	This command is used to add a new segment waveform
COMMAND SYNTAX	<channel>:SEQuence:SEGMent:Add <channel>:={C1,C2}.
EXAMPLE	Add a segment waveform for the sequence of channel 1 *:C1:SEQuence:SEGMent:Add*

3.13.20 <channel>:SEQuence:SEGMent<x>:INSErt

DESCRIPTION	This command is used to insert a new segment after a certain segment in the sequence waveform
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:INSErt <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Insert a new segment after the third segment of the channel 1 sequence *:C1:SEQuence:SEGMent3:INSErt*

3.13.21 <channel>:SEQuence:SEGMent<x>:DELEte

DESCRIPTION	This command is used to delete a segment in the sequence waveform
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:DELEte <channel>:={C1,C2}.

	<x>:= segment number
EXAMPLE	Delete the waveform of the third segment in the channel 1 sequence *C1:SEQuence:SEGMent3:DELEte*

3.13.22 <channel>:SEQuence:SEGMent:Clear

DESCRIPTION	This command is used to clear the segments of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent:Clear <channel>:={C1,C2}.
EXAMPLE	Clear the segment of channel 1 *:C1:SEQuence:SEGMent:Clear*

3.13.23 <channel>:SEQuence:SEGMent<x>:GOTO

DESCRIPTION	This command is used to set (query) the jump of a segment in the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:GOTO <num> <channel>:={C1,C2}. <x>:= segment number <num>:= segment number
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:GOTO? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the jump number of the first segment of the channel 1 sequence to 3 *:C1:SEQuence:SEGMent1:GOTO 3* Query the jump number of the first segment of the channel 1 sequence *:C1:SEQuence:SEGMent1:GOTO?* Return: *3*

3.13.24 <channel>:SEQuence:SEGMent<x>:LENGth

DESCRIPTION	This command is used to set (query) the length of the waveform of a certain segment of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:LENGth <len>

	<channel>:={C1,C2}. <x>:= segment number <len>:= It will be automatically truncated to an integer multiple of 16
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:LENGth? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the length of the waveform of the third segment of the channel 1 sequence to 16384 *:C1:SEQuence:SEGMent3:LENGth 16384* Query the length of the waveform of the third segment of the channel 1 sequence *:C1:SEQuence:SEGMent3:LENGth?* Return: *16384*

3.13.25 <channel>:SEQuence:SEGMent<x>:REPeat:COUNt

DESCRIPTION	This command is used to set (query) the repetition times of a certain segment waveform of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:REPeat:COUNt <count> <channel>:={C1,C2}. <x>:=segment number <count>:= The maximum value is related to the waveform length of this segment and the total length of other segments.
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:REPeat:COUNt? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the waveform of the third segment of the channel 1 sequence to repeat twice *:C1:SEQuence:SEGMent3:REPeat:COUNt 2* Query the waveform repetition times of the third segment of channel 1 sequence *:C1:SEQuence:SEGMent3:REPeat:COUNt?* Return: *2*

3.13.26 <channel>:SEQuence:SEGMent<x>:WAVeform

DESCRIPTION	This command is used to set (query) the waveform of a certain segment of the sequence through the waveform name
COMMAND SYNTAX	[:SOURce]:<channel>:SEQuence:SEGMent<x>:WAVeform <name> <channel>:={C1,C2}. <x>:=segment number <name>:=waveform name. See section 3.9.1 for available waveform names
QUERY SYNTAX	[:SOURce]:<channel>:SEQuence:SEGMent<x>:WAVeform? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the waveform of the third segment of channel 1 sequence as sine *:C1:SEQuence:SEGMent3:WAVeform sine* Query the waveform of the third segment of channel 1 sequence (return the waveform name) *:C1:SEQuence:SEGMent3:WAVeform?* Return: *sine*

3.13.27 <channel>:SEQuence:SEGMent<x>:AMPlitude

DESCRIPTION	This command is used to set (query) the waveform amplitude of a certain segment of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:AMPlitude <amp> <channel>:={C1,C2}. <x>:= segment number <amp>:={0 to 20Vpp}
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:AMPlitude? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the amplitude of the third segment waveform of channel 1 sequence to 2Vpp *:C1:SEQuence:SEGMent3:AMPlitude 2* Query the amplitude of the waveform of the third segment of channel 1 sequence *:C1:SEQuence:SEGMent3:AMPlitude?*

	Return: *2*
NOTE	segment number = {1 to 1024}

3.13.28 <channel>:SEQuence:SEGMent<x>:OFFset

DESCRIPTION	This command is used to set (query) the waveform offset of a certain segment of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:OFFset <offset> <channel>:={C1,C2}. <x>:= segment number <offset>:={-8V to 8V}
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:OFFset? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the offset of the third segment waveform of channel 1 sequence to 2Vdc *:C1:SEQuence:SEGMent3:OFFset 2* Query the offset of the third segment waveform of channel 1 sequence *:C1:SEQuence:SEGMent3:OFFset?* Return*:* *2*

3.13.29 <channel>:SEQuence:SEGMent<x>:VOLTage:HIGH

DESCRIPTION	This command is used to set (query) the high-level value of the waveform of a certain segment of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:VOLTage:HIGH <highLevel> <channel>:={C1,C2}. <x>:= segment number <highLevel>:={lowLevel to 12V}

QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:VOLTage:HIGH? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the high level of the third segment waveform of channel 1 sequence to 8V *:C1:SEQuence:SEGMent3:VOLTage:HIGH 8* Query the high-level value of the waveform of the third segment of the channel 1 sequence *:C1:SEQuence:SEGMent3:VOLTage:HIGH?* Return: *8*
NOTE	segment number = {1 to 1024}

3.13.30 <channel>:SEQuence:SEGMent<x>:VOLTage: LOW

DESCRIPTION	This command is used to set (query) the low-level value of the waveform of a certain segment of the sequence
COMMAND SYNTAX	<channel>:SEQuence:SEGMent<x>:VOLTage:LOW <lowLevel> <channel>:={C1,C2}. <x>:= segment number <lowLevel>:={-12V to highLevel}
QUERY SYNTAX	<channel>:SEQuence:SEGMent<x>:VOLTage:LOW? <channel>:={C1,C2}. <x>:= segment number
EXAMPLE	Set the low level of the third segment waveform of channel 1 sequence to 1V *:C1:SEQuence:SEGMent3:VOLTage:LOW 1* Query the low-level value of the third segment waveform of channel 1 sequence *:C1:SEQuence:SEGMent3:VOLTage:LOW?* Return: *1*

3.14 IQ Command

3.14.1 IQ:WAVeinfo?

DESCRIPTION	This command queries I/Q waveform information.
QUERY SYNTAX	IQ:WAVeinfo?
EXAMPLE	Query current I/Q waveform information: *IQ:WAVeinfo?* Return: *WAVE_INFO,SYMBOL_LENTH,1024,OVER_SAMPLING,4,MODULATION,2ASK,FILTER_TYPE,RootCosine,FILTER_ALPHA,0.35*

3.14.2 IQ:CENTerfreq

DESCRIPTION	This command sets the center frequency of I/Q modulation.
COMMAND SYNTAX	IQ:CENTerfreq <center freq><unit> <center freq>:= center frequency. Please refer to the data sheet for the valid range of this parameter. <unit>:={Hz, kHz, MHz, GHz}. The default unit is "Hz".
QUERY SYNTAX	IQ:CENTerfreq?
RESPONSE FORMAT	<center freq>(Expressed in Hz)
EXAMPLE	Set the center frequency to 1kHz: *IQ:CENTerfreq 1000Hz*

3.14.3 IQ:SAMPlerate

DESCRIPTION	This command sets the I/Q sampling rate.
COMMAND SYNTAX	IQ:SAMPlerate <samp rate><unit> <samp rate>:= Sampling rate. Please refer to the data sheet for the valid range of this parameter. <unit>:={Hz, kHz, MHz, GHz}. The default unit is "Hz".
QUERY SYNTAX	IQ:SAMPlerate?
RESPONSE FORMAT	<samp rate>(Expressed in Hz)
EXAMPLE	Set sample rate to 100kHz: *IQ:SAMPlerate 100000* or *IQ:SAMP 100kHz*

3.14.4 IQ:SYMBolrate

DESCRIPTION	This command is used to set the I/Q symbol rate.
COMMAND SYNTAX	IQ:SYMBolrate <symbol rate><unit> <symbol rate>:= Symbol rate. Please refer to the data sheet for the valid range of this parameter. <unit>:={S/s, KS/s, MS/s}. The default unit is "S/s".
QUERY SYNTAX	IQ:SYMBolrate?
RESPONSE FORMAT	<symbol rate>(Expressed in S/s)
EXAMPLE	Set symbol rate to 1MS/s: *IQ:SYMB 1MS/s*

3.14.5 IQ:AMPLitude

DESCRIPTION	This command sets the I/Q amplitude.
COMMAND SYNTAX	IQ:AMPLitude <amplitude><unit> <amplitude>:= Amplitude. Please refer to the data sheet for the valid range of this parameter. <unit>:={Vrms, mVrms, dBm}. The default unit is "Vrms".
QUERY SYNTAX	IQ:AMPLitude?
RESPONSE FORMAT	<amplitude>(Expressed in Vrms)
EXAMPLE	Set the amplitude of IQ (sqrt(I2+Q2)) to 0.2Vrms: *IQ:AMPL 0.2*

3.14.6 IQ:IQADjustment:GAIN

DESCRIPTION	This command adjusts the ratio of I and Q while maintaining the IQ composite state.
COMMAND SYNTAX	IQ:IQADjustment:GAIN <gain ratio><unit> <gain ratio>:= I to Q gain ratio. <unit>:={dB}.
QUERY SYNTAX	IQ:IQADjustment:GAIN?
RESPONSE FORMAT	<gain ratio>(Expressed in dB)
EXAMPLE	Set the I/Q gain ratio to 0.1dB: *IQ:IQADjustment:GAIN 0.1*

3.14.7 IQ:IQADjustment:IOFFset

DESCRIPTION	This command adjusts the offset of the I channel.
COMMAND SYNTAX	IQ:IQADjustment:IOFFset <offset><unit> <offset>:= offset of I. <unit>:={V, mV, uV}. The default unit is "V".
QUERY SYNTAX	IQ:IQADjustment:IOFFset?
RESPONSE FORMAT	<offset>(Expressed in V)
EXAMPLE	Set the bias of I to 1mV: *IQ:IQADjustment:IOFFset 1mV*

3.14.8 IQ:IQADjustment:QOFFset

DESCRIPTION	This command adjusts the offset of the Q channel.
COMMAND SYNTAX	IQ:IQADjustment:QOFFset <offset><unit> <offset>:= offset of Q. <unit>:={V, mV, uV}. The default unit is "V".
QUERY SYNTAX	IQ:IQADjustment:QOFFset?
RESPONSE FORMAT	<offset>(Expressed in V)
EXAMPLE	Set the bias of Q to -1mV: *IQ:IQADjustment:QOFFset -0.001*

3.14.9 IQ:IQADjustment:QSKew

DESCRIPTION	This command adjusts the phase angle of the I and Q vectors by increasing or decreasing the phase angle of Q.
COMMAND SYNTAX	IQ:IQADjustment:QSKew <angle> <angle>:= angle. The unit is °.
QUERY SYNTAX	IQ:IQADjustment:QSKew?
RESPONSE FORMAT	<angle>(Expressed in °)
EXAMPLE	Set the Q angle to 1°: *IQ:IQADjustment:QSKew 1.0*

3.14.10 IQ:TRIGger:SOURce

DESCRIPTION	This command sets the trigger source of I/Q.
COMMAND SYNTAX	IQ:TRIGger:SOURce <source> <source>:={INTernal,EXTernal,MANual,Timer}. Among them, INTernal is an internal trigger, EXTernal is an external trigger, MANual is a manual trigger, and Timer is a timer trigger.
QUERY SYNTAX	IQ:TRIGger:SOURce?
RESPONSE FORMAT	<source>
EXAMPLE	Set the trigger source to internal trigger: *IQ:TRIGger:SOURce INTernal*

3.14.11 IQ:WAVEload:BUILtin

DESCRIPTION	This command is used to select I/Q waveforms from the built-in waveform list.
COMMAND SYNTAX	IQ:WAVEload:BUILtin <name> <name>:={Waveform names in the table below}.
QUERY SYNTAX	IQ:WAVEload?
RESPONSE FORMAT	BUILtin|USERstored <name>
EXAMPLE	Set the I/Q waveform to 2ASK of the built-in waveform: *IQ:WAVE:BUIL "2ASK"*

2ASK	4ASK	8ASK	BPSK	QPSK
8PSK	DBPSK	DQPSK	D8PSK	8QAM
16QAM	32QAM	64QAM	128QAM	256QAM

3.14.12 IQ:WAVEload:USERstored

DESCRIPTION	This command is used to select I/Q waveforms in user stored waveforms.
COMMAND SYNTAX	Format1: IQ:WAVEload:USERstored <name> <name>:={Waveforms from user storage}. Format2: IQ:WAVEload:USERstored <path> <path>:={Waveform path from user storage (local, network storage, USB flash drive), including file name and suffix}.

QUERY SYNTAX	IQ:WAVEload?
RESPONSE FORMAT	BUILtin|USERstored <name>
EXAMPLE	Set the I/Q waveform to user storage waveform wave1.arb: *IQ:WAVEload:USERstored wave1.arb* Set the I/Q waveform to the U disk storage waveform wave1.arb: *IQ:WAVEload:USERstored "U-disk0/ wave/wave1.arb"*

3.14.13 IQ:FrequencySampling

DESCRIPTION	This command sets the I/Q sampling rate.
COMMAND SYNTAX	IQ:FrequencySampling <sampling> <sampling>:={1000-200000000}. The unit is "Hz".
QUERY SYNTAX	Query the current sampling rate: IQ:FrequencySampling? Query the settable sampling rate range: IQ:FrequencySamplingLimit?
RESPONSE FORMAT	<sampling> MAX,<max_sampling>,MIN,<min_sampling>
EXAMPLE	Set I/Q sampling rate to 2000000 Hz: *IQ:FrequencySampling 2000000*

3.15 Channel Track/Coupling Command

DESCRIPTION	This command sets or gets the channel coupling parameters. Only when TRACE is set to OFF, the other coupling parameters can be set.
COMMAND SYNTAX	COUPling <parameter>,<value> <parameter>:= {a parameter from the table below}. <value>:={value of the corresponding parameter}.

Parameters	Value	Description
TRACE	<track_enble>	:= {ON, OFF} State of channel tracking.
FCOUP	<fcoup>	:= {ON, OFF} State of frequency coupling
FDEV	<frq_dev>	:= frequency deviation between the 2 channels. The unit is Hertz “Hz”.
FRAT	<frat>	:= frequency ratio between the 2 channels.
PCOUP	<pcoup>	:={ON, OFF} State of phase coupling
PDEV	<pha_dev>	:= phase deviation between the 2 channels. The unit is degree “°”.
PRAT	<prat>	:= phase ratio between the 2 channels.
ACOUP	<acoup>	:= {ON, OFF},State of amplitude coupling
ARAT	<arat>	:= amplitude ratio between the 2 channels.
ADEV	<adev>	:= amplitude deviation between the 2 channels. The unit is volts, peak-to-peak “Vpp”.
TRDUCH	<trigger_ch>	:= {ON, OFF}, Set trigger channel, ON means double channel trigger, and OFF means single channel trigger.

QUERY SYNTAX	COUPling?
RESPONSE FORMAT	COUP <parameter> <parameter>:= { All parameters of coupling}.
EXAMPLE	Set coupling state to ON: *COUP STATE,ON* Set frequency coupling state to ON: *COUP FCOUP,ON* Set frequency deviation to 5 Hz: *COUP FDEV,5*

	Query coupling information. *COUP?* Return: *COUP TRACE,OFF,FCOUP,ON,PCOUP,ON,ACOUP,ON,FDEV,5HZ,PRAT,1,ARAT,2*

3.16 Channel Copy Command

DESCRIPTION	This command copies parameters from one channel to another.
COMMAND SYNTAX	ParaCoPy <destination_channel>,<src_channel> < destination_channel>:= {C1, C2}. <src_channel>:= {C1, C2}. Note: the parameters C1 and C2 must be set to the device together.
EXAMPLE	Copy parameters from CH1 to CH2. *PACP C2,C1*

3.17 Invert Command

DESCRIPTION	This command sets or gets the polarity of specified channel.
COMMAND SYNTAX	<channel>:INVerT <state> <channel>:= {C1, C2}. <state>:= {ON, OFF}.
QUERY SYNTAX	<channel>:INVerT? <channel>:={C1, C2}.
RESPONSE FORMAT	<channel>:INVT <state>
EXAMPLE	Set CH1 polarity to invert *C1:INVT ON* Read the polarity of CH1. *C1:INVT?* Return: *C1:INVT ON*

3.18 Equal Phase Command

DESCRIPTION	This command is used to set the phase synchronization of two channels
COMMAND SYNTAX	EQPHASE
RESPONSE FORMAT	EQPHASE <state>
EXAMPLE	*EQPHASE*

3.19 Waveform Combining Command

DESCRIPTION	This command sets or gets the waveform combining parameters.
COMMAND SYNTAX	<channel>:CoMBiNe <state> <channel>:= {C1, C2}. <state>:= {ON, OFF}.
QUERY SYNTAX	<channel>:CoMBiNe? <channel>:= {C1, C2}.
RESPONSE FORMAT	<channel>:CMBN <state>
EXAMPLES	Enable waveform combining forf CH1: *C1:CMBN ON* Query the waveform combining state of CH2: *C2:CMBN?* Return: *C2:CMBN OFF*

3.20 Power-On State command

DESCRIPTION	This command sets or gets the state of the channel output after power on.
COMMAND SYNTAX	<channel>:output POWERON_STATE,<state> <channel>:= {C1, C2}. <state>:= {ON, OFF}.
QUERY SYNTAX	<channel>:output? <channel>:= {C1, C2}.
RESPONSE FORMAT	<channel>:output?
EXAMPLES	Set the output status of channel 1 to on after power on: *C1:output POWERON_STATE,ON*

	Query that the output status of channel 1 is on after power on: *C1:output?* Return: *C1:OUTP ON,LOAD,HZ,POWERON_STATE,OFF,PLRT,NOR*

3.21 Sync Command

DESCRIPTION	This command sets the synchronization signal.
COMMAND SYNTAX	<channel>:SYNC <state> <channel>:= {C1, C2}. <state>:= {ON, OFF}. SYNC TYPE, <TYPE> <TYPE>:={CH1,CH2,MOD_CH1,MOD_CH2}.
QUERY SYNTAX	<channel>:SYNC? <channel>:= {C1, C2}.
RESPONSE FORMAT	<channel>:SYNC <state>
EXAMPLE	Turn on sync output and set the source as modulating signal of CH1: *C1:SYNC ON,TYPE,MOD_CH1* Read the state of CH1 sync. *C1:SYNC?* Return: *C1:SYNC ON,TYPE,MOD_CH1*

DESCRIPTION	This command is used to set the synchronization signal output type.
COMMAND SYNTAX	<channel>:SYNC:<parameter> <value> <channel>:= {C1, C2}. <parameter>:= OUT_MODE <value>:={Pulse, Square}.
EXAMPLE	Set the output mode of CH1 synchronous signal as 50% duty square wave: *C1:SYNC:OUT_MODE Square*

3.22 Clock Source Command

DESCRIPTION	This command sets or gets the clock source.
COMMAND SYNTAX	ROSCillator <src> <src>:= {INT, EXT} ROSCillator 10MOUT,<state> <state>:={ ON,OFF }
QUERY SYNTAX	ROSC?
RESPONSE FORMAT	ROSC <src>,10MOUT,<state>
EXAMPLE	Set internal time base as the source: *ROSC INT* Enable 10MHz output: *ROSC 10MOUT,ON*

3.23 Phase Mode Command

DESCRIPTION	This command sets or gets the phase mode.
COMMAND SYNTAX	MODE <parameter> <parameter>:= {PHASE-LOCKED, INDEPENDENT}.
QUERY SYNTAX	MODE?
RESPONSE FORMAT	MODE <parameter>
EXAMPLE	Set the phase mode to INDEPENDENT: *MODE INDEPENDENT*

3.24 Freq_Com State Command

DESCRIPTION	This command sets or gets the Frequency compensation switch status.
COMMAND SYNTAX	Freq_com_switch <state> <state>={ON , OFF}.
QUERY SYNTAX	Freq_com_switch?
RESPONSE FORMAT	Freq_com_switch <state>
EXAMPLE	Set the Frequency compensation switch status. to ON: *Freq_com_switch ON*

3.25 PowerOn Setting Command

DESCRIPTION	This command sets or gets the power-on system setting.
COMMAND SYNTAX	Format1: Sys_CFG <mode> <mode>:= {DEFAULT, LAST,USER} Format2: Sys_CFG<config><filepath> <config>:={USER, PRESET} <filepath>:= { The path of the configuration file stored by the user (local, network storage, USB flash disk), including the file name and suffix }
QUERY SYNTAX	Sys_CFG?
RESPONSE FORMAT	SCFG <mode> Sys_CFG<config><filepath>
EXAMPLE	Set the power-on system setting to LAST: *SCFG LAST* Set boot recovery file: *SCFG USER,"state.xml"*

3.26 Multi-Device Sync

DESCRIPTION	This command set up synchronization between two or more instruments and achieve in-phase output
COMMAND SYNTAX	CASCADE < parameter >,< value > < value >:= { The values of relevant parameters }

parameter	value	describe
STATE	<state>	:={ON,OFF} Turn on or off multi device synchronization
MODE	<type>	:= {MASTER，SLAVE}
DELAY	<time>	:= {0-0.000025},UNIT=s，This parameter can only be set in slave mode
SYNC_PHASE		Set once and trigger the "Sync Device" button once

QUERY SYNTAX	CASCADE?
EXAMPLE	Set the device as slave and the delay to 0.0000001s: *CASCADE STATE,ON,MODE,SLAVE,DELAY,0.0000001*

3.27 Number Format Command

DESCRIPTION	This command sets or gets the number format.
COMMAND SYNTAX	NumBer_ForMat PNT,<pnt>, NumBer_ForMa SEPT,<sept> <pnt>:= {Dot, Comma}. The point format. <sept>:= {Space, Off, On}. The separator format.
QUERY SYNTAX	NBFM?
RESPONSE FORMAT	NBFM PNT,<pnt>, SEPT,<sept>
EXAMPLE	Set point format to DOT: *NBFM PNT,DOT* Set separator format to ON: *NBFM SEPT,ON* Read the number format: *NBFM?* Return: *NBFM PNT, DOT, SEPT, ON*

3.28 Language Command

DESCRIPTION	This command sets or gets the system language.
COMMAND SYNTAX	LAnGuaGe <language> <language>:= {EN,CH}, where EN is English, CH is Chinese
QUERY SYNTAX	LAnGuaGe?
RESPONSE FORMAT	LAGG <language>
EXAMPLE	Set language to English: *LAGG EN* Read language *LAGG?* Return: *LAGG EN*

3.29 Buzzer Command

DESCRIPTION	This command turns on or off the buzzer.
COMMAND SYNTAX	BUZZer <state> <state>:= {ON, OFF}.

QUERY SYNTAX	BUZZer?
RESPONSE FORMAT	BUZZ <state>
EXAMPLE	Turn on the buzzer: *BUZZ ON*

3.30 SCPI update UI Command

DESCRIPTION	This command is used to turn on or off the status of SCPI update UI.
COMMAND SYNTAX	SCPI_UPDATE_UI <state> <state>:= {ON, OFF}.
QUERY SYNTAX	SCPI_UPDATE_UI?
RESPONSE FORMAT	<state>
EXAMPLE	Set SCPI update UI status off: *SCPI_UPDATE_UI OFF*

3.31 Screen capture Command

DESCRIPTION	This command is used to set the picture format of the screenshot.
COMMAND SYNTAX	SCREEN_SHOT_TYPE <format> <format>:= {PNG, BMP}.
QUERY SYNTAX	SCREEN_SHOT_TYPE?
RESPONSE FORMAT	<format>
EXAMPLE	Set the screenshot format to BMP format: *SCREEN_SHOT_TYPE BMP*

DESCRIPTION	Set screen shots.
COMMAND SYNTAX	SCREEN_SHOT

3.32 Screen Saver Command

DESCRIPTION	This command sets or gets the screen saver time. The unit is minutes “m”.
COMMAND SYNTAX	SCreen_SaVe <parameter> <parameter>:= {OFF, 1, 5, 15, 30, 60, 120, 300}.
QUERY SYNTAX	SCreen_SaVe?

RESPONSE FORMAT	SCSV <parameter>
EXAMPLE	Set screen saver time to 5 minutes: *SCSV 5* Read the current screen saver time: *SCreen_SaVe?* Return: *SCSV 5MIN*

3.33 Frequency Counter Command

DESCRIPTION	This command sets or gets the frequency counter parameters.
COMMAND SYNTAX	FreqCouNTer <parameter>,<value> <parameter>:= {a parameter from the table below}. <value>:= {value of the corresponding parameter}.

Parameters	Value	Description
STATE	<state>	:={ON, OFF} State of frequency counter.
FRQ	<frequency>	Measured frequency. The unit is Hertz “Hz”. Can’t be set.
PW	<pos_width>	Measured positive width. The unit is seconds “s”. Can’t be set.
NW	<neg_width>	Measured negative width. The unit is seconds “s”. Can’t be set.
DUTY	<duty>	Measured duty cycle. The unit is "%". Can’t be set.
FRQDEV	<freq_dev>	Measured frequency deviation. The unit is "ppm". Can’t be set.
REFQ	<ref_freq>	Expected frequency, for calculating the frequency deviation. The unit is Hertz “Hz”.
TRG	<triglev>	Trigger level. The range of valid values depends on the model. The unit is volts “V”.
MODE	<mode>	:={AC, DC} Coupling mode.
HFR	<HFR>	:={ON, OFF} State of High Frequency Rejection.
DEF	NA	Restore counter to default settings

QUERY SYNTAX	FreqCouNTer?
RESPONSE FORMAT	FCNT <parameter> <parameter>:={All parameters of the frequency counter}
EXAMPLE	Turn frequency counter on: *FCNT STATE,ON* Set reference frequency to 1000 Hz: *FCNT REFQ,1000* Query frequency counter information: *FCNT?* Return: *FCNT STATE,ON,FRQ,10000000HZ,DUTY,59.8568,REFQ,1e+07HZ,TRG,0V,PW,5.98568e-08S,NW,4.01432e-08S,FRQDEV,0ppm,MODE,AC,HFR,OFF*
DESCRIPTION	Clear the statistics of the counter
COMMAND SYNTAX	*Clear_fcnt*

3.34 Over-Voltage Protection Command

DESCRIPTION	This command sets or gets the state of over-voltage protection.
COMMAND SYNTAX	VOLTPRT <state> <state>:= {ON, OFF}
QUERY SYNTAX	VOLTPRT?
RESPONSE FORMAT	VOLTPRT <state>

3.35 Store List Command

DESCRIPTION	This command is used to read the stored waveforms list with indexes and names. If the storage unit is empty, the command will return “EMPTY”
QUERY SYNTAX	Format1: SToreList? Format2: SToreList? <parameter> < parameter >=:{ Parameters in the following table } Format3: SToreList? <USER>,<path> < path >=:{ Specify the path of network storage or USB flash disk, as shown in the table below }
RESPONSE FORMAT	<waveform name>

parameter		function
BUILDIN	< parameter >	Query built-in waveform name and index number
USER	< parameter >	Query locally saved waveform name

EXAMPLE

(1) Read all arbitrary wave names saved in the device (excluding network storage and data in USB flash disk):

STL?

Return:

STL M0, sine, M1, noise, M2, stairup, M3, stairdn, M4, stairud, M5, ppulse, M6, npulse, M7, trapezia, M8, upramp, M9, dnramp, M10, exp_fall, M11, exp_rise, M12, logfall, M13, logrise, M14, sqrt, M15, root3, M16, x^2, M17, x^3, M18, sinc, M19, gaussian, M20, dlorentz, M21, haversine, M22, lorentz, M23, gauspuls, M24, gmonopuls, M25, tripuls, M26, cardiac, M27, quake, M28, chirp, M29, twotone, M30, snr, M31, EMPTY, M32, EMPTY, M33, EMPTY, M34, hamming, M35, hanning, M36, kaiser, M37, blackman, M38, gaussiwin, M39, triangle, M40, blackmanharris, M41, bartlett, M42, tan, M43, cot, M44, sec, M45, csc, M46, asin, M47, acos, M48, atan, M49, acot, M50, EMPTY, M51, EMPTY, M52, EMPTY, M53, DDROPOUT, M54, FCLK1, M55, FSDA1, M56, EMPTY, M57, EMPTY, M58, EMPTY, M59, EMPTY

(2) Read the built-in waveform name saved in the device:

STL? BUILDIN

Return:

STL M0,sine, M1, noise, M10, ExpFal, M100, ECG14, M101, ECG15, M102, LFPulse, M103, Tens1, M104, Tens2, M105, Tens3, M106, Airy, M107, Besselj, M108, Bessely, M109, Dirichlet, M11, ExpRise, M110, Erf, M111, Erfc, M112, ErfcInv, M113, ErfInv, M114, Laguerre, M115, Legend, M116, Versiera, M117, Weibull, M118, LogNormal, M119, Laplace, M12, LogFall, M120, Maxwell, M121, Rayleigh, M122, Cauchy, M123, CosH, M124, CosInt, M125, CotH, M126, CscH, M127, SecH, M128, SinH, M129, SinInt, M13, LogRise, M130, TanH, M131, ACosH, M132, ASecH, M133, ASinH, M134, ATanH, M135, ACsch, M136, ACoth, M137, Bartlett, M138, BohmanWin, M139, ChebWin, M14, Sqrt, M140, FlattopWin, M141, ParzenWin, M142, TaylorWin, M143, TukeyWin, M144, SquareDuty01, M145, SquareDuty02, M146, SquareDuty04, M147, SquareDuty06, M148, SquareDuty08, M149, SquareDuty10, M15, Root3, M150, SquareDuty12, M151, SquareDuty14, M152, SquareDuty16, M153, SquareDuty18, M154, SquareDuty20, M155, SquareDuty22, M156, SquareDuty24, M157,

SquareDuty26, M158, SquareDuty28, M159, SquareDuty30, M16, X^2, M160, SquareDuty32, M161, SquareDuty34, M162, SquareDuty36, M163, SquareDuty38, M164, SquareDuty40, M165, SquareDuty42, M166, SquareDuty44, M167, SquareDuty46, M168, SquareDuty48, M169, SquareDuty50, M17, X^3, M170, SquareDuty52, M171, SquareDuty54, M172, SquareDuty56, M173, SquareDuty58, M174, SquareDuty60, M175, SquareDuty62, M176, SquareDuty64, M177, SquareDuty66, M178, SquareDuty68, M179, SquareDuty70, M18, Sinc, M180, SquareDuty72, M181, SquareDuty74, M182, SquareDuty76, M183, SquareDuty78, M184, SquareDuty80, M185, SquareDuty82, M186, SquareDuty84, M187, SquareDuty86, M188, SquareDuty88, M189, SquareDuty90, M19, Gaussian, M190, SquareDuty92, M191, SquareDuty94, M192, SquareDuty96, M193, SquareDuty98, M194, SquareDuty99, M195, demo1_375pts, M196, demo1_16kpts, M197, demo2_3kpts, M198, demo2_16kpts, M2, StairUp, M20, Dlorentz, M21, Haversine, M22, Lorentz, M23, Gauspuls, M24, Gmonopuls, M25, Tripuls, M26, Cardiac, M27, Quake, M28, Chirp, M29, Twotone, M3, StairDn, M30, SNR, M31, Hamming, M32, Hanning, M33, kaiser, M34, Blackman, M35, Gausswin, M36, Triangle, M37, Bartlett-Hann, M38, Bartlett, M39, Tan, M4, StairUD, M40, Cot, M41, Sec, M42, Csc, M43, Asin, M44, Acos, M45, Atan, M46, Acot, M47, Square, M48, SineTra, M49, SineVer, M5, Ppulse, M50, AmpALT, M51, AttALT, M52, RoundHalf, M53, RoundsPM, M54, BlaseiWave, M55, DampedOsc, M56, SwingOsc, M57, Discharge, M58, Pahcur, M59, Combin, M6, Npulse, M60, SCR, M61, Butterworth, M62, Chebyshev1, M63, Chebyshev2, M64, TV, M65, Voice, M66, Surge, M67, Radar, M68, Ripple, M69, Gamma, M7, Trapezia, M70, StepResp, M71, BandLimited, M72, CPulse, M73, CWPulse, M74, GateVibr, M75, LFMPulse, M76, MCNoise, M77, AM, M78, FM, M79, PFM, M8, Upramp, M80, PM, M81, PWM, M82, EOG, M83, EEG, M84, EMG, M85, Pulseilogram, M86, ResSpeed, M87, ECG1, M88, ECG2, M89, ECG3, M9, Dnramp, M90, ECG4, M91, ECG5, M92, ECG6, M93, ECG7, M94, ECG8, M95, ECG9, M96, ECG10, M97, ECG11, M98, ECG12, M99, ECG13

(3) Read user-defined waveform name from the device::

STL? USER

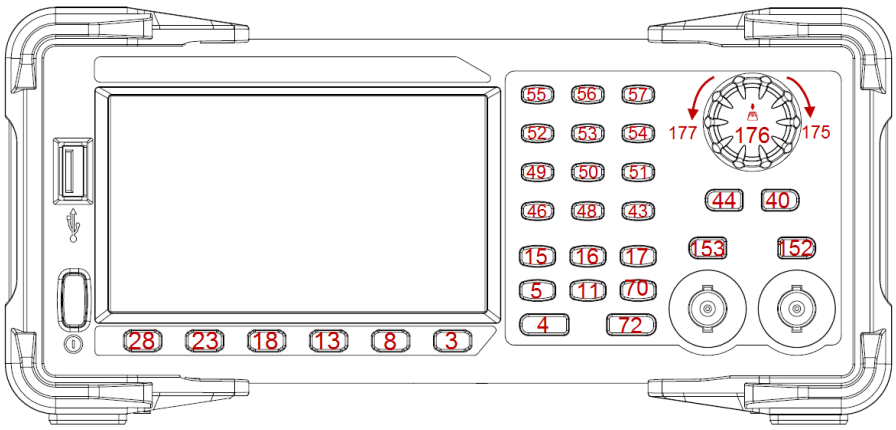
Return:

STL WVNM,sinc_8M,sinc_3000000,sinc_1664000,ramp_8M, sinc_2000000,sinc_50000,square_8M,sinc_5000,wave1,square_1M

3.36 Virtual Key Command

DESCRIPTION	This command is used to simulate pressing a key on the front panel.
COMMAND SYNTAX	VirtualKEY VALUE,<value>,STATE,<state> <value>:= {a Name or Index of the virtual keys from the table below}. <state>:= {0,1}, where 1 is effective to virtual value, and 0 is useless.

Name	Indexes	Name	Indexes
KB_FUNC1	28	KB_NUMBER_4	52
KB_FUNC2	23	KB_NUMBER_5	53
KB_FUNC3	18	KB_NUMBER_6	54
KB_FUNC4	13	KB_NUMBER_7	55
KB_FUNC5	8	KB_NUMBER_8	56
KB_FUNC6	3	KB_NUMBER_9	57
KB_MOD	15	KB_POINT	46
KB_SWEEP	16	KB_NEGATIVE	43
KB_BURST	17	KB_LEFT	44
KB_UTILITY	11	KB_RIGHT	40
KB_PARAMETER	5	KB_OUTPUT1	153
KB_NUMBER_0	48	KB_OUTPUT2	152
KB_NUMBER_1	49	KB_KNOB_RIGHT	175
KB_NUMBER_2	50	KB_KNOB_LEFT	177
KB_NUMBER_3	51	KB_KNOB_DOWN	176



Keys and Indices on the SDG6000X

EXAMPLE	*VKEY VALUE,15,STATE,1* *VKEY VALUE,KB_SWEEP,STATE,1*

3.37 IP Command

DESCRIPTION	This command sets and gets the system IP address.
COMMAND SYNTAX	SYSTem:COMMunicate:LAN:IPADdress “<parameter1>.<parameter2>.<parameter3>.<parameter4>” <parameter1>:={an integer value between 1 and 223}. <parameter2>:={an integer value between 0 and 255}. <parameter3>:={an integer value between 0 and 255}. <parameter4>:={an integer value between 0 and 255}.
QUERY SYNTAX	SYSTem:COMMunicate:LAN:IPADdress?
EXAMPLES	Set IP address to 10.11.13.203: *SYST:COMM:LAN:IPAD “10.11.13.203”* Get the IP address: *SYST:COMM:LAN:IPAD?* Return: *“10.11.13.203”*

3.38 Subnet Mask Command

DESCRIPTION	This command sets and gets the system subnet mask.
COMMAND SYNTAX	SYSTem:COMMunicate:LAN:SMASk “<parameter1>.<parameter2>.<parameter3>.<parameter4>” <parameter1>:={an integer value between 0 and 255}. <parameter2>:={an integer value between 0 and 255}. <parameter3>:={an integer value between 0 and 255}. <parameter4>:={an integer value between 0 and 255}.
QUERY SYNTAX	SYSTem:COMMunicate:LAN:SMASk?
EXAMPLES	Set the subnet mask to 255.0.0.0: *SYST:COMM:LAN:SMAS “255.0.0.0”* Get the subnet mask: *SYST:COMM:LAN:SMAS?* Return: *“255.0.0.0”*

3.39 Gateway Command

DESCRIPTION	This command sets and gets the system gateway.
COMMAND SYNTAX	SYSTem:COMMunicate:LAN:GATeway “<parameter1>.<parameter2>.<parameter3>.<parameter4>” <parameter1>:={an integer value between 0 and 223}. <parameter2>:={an integer value between 0 and 255}. <parameter3>:={an integer value between 0 and 255}. <parameter4>:={an integer value between 0 and 255}.
QUERY SYNTAX	SYSTem:COMMunicate:LAN:GATeway?
EXAMPLES	Set Gateway to 10.11.13.5: *SYSTem:COMMunicate:LAN:GATeway “10.11.13.5”* Get gateway: *SYSTem:COMMunicate:LAN:GATeway?* Return: *“10.11.13.5”*

3.40 Gpib Command

DESCRIPTION	This command sets and gets the port number for Gpib.
COMMAND SYNTAX	SYSTem:COMMunicate:GPIB:ADDRess <num>
QUERY SYNTAX	SYSTem:COMMunicate:GPIB:ADDRess?
EXAMPLES	Set the port number of Gpib to 19: *SYSTem:COMMunicate:GPIB:ADDRess 19* Get the port number of Gpib: *SYSTem:COMMunicate:GPIB:ADDRess?* Return: *19*

3.41 File Operation Commands

3.41.1 MMEMory:DELete

DESCRIPTION	This command deletes the file.
COMMAND SYNTAX	MMEMory:DELete <parameter> <parameter>:=”The path of the file”.
EXAMPLE	Delete a file whose path is “Local/1000pts.bin”: *MMEMory:DELete “1000pts.bin”*

3.41.2 MMEMory:RDIRectory

DESCRIPTION	This command deletes the directory.
COMMAND SYNTAX	MMEMory:RDIRectory <parameter> <parameter>:=”The path of the directory”.
EXAMPLE	Delete a directory whose path is “Local/test”: *MMEMory:RDIRectory “test”*

3.41.3 MMEMory:MDIRectory

DESCRIPTION	This command creates a new directory.
COMMAND SYNTAX	MMEMory:MDIRectory <parameter> <parameter>:=”The path of the directory”.
EXAMPLE	Create a directory whose path is “Local/test”: *MMEMory:MDIRectory “test”*

3.41.4 MMEMory:CATalog

DESCRIPTION	This command checks the files and the directories from the path or checks the file with specific type.
QUERY SYNTAX	MMEMory:CATalog? <parameter> <parameter>:=”The path of the directory”. MMEMory:CATalog:DATA:ARBitrary? <parameter> <parameter>:=”The path of the directory”. MMEMory:CATalog:STATe:XMLanguage? <parameter> <parameter>:=”The path of the directory”.
RESPONSE FORMAT	remain space, used space

	“File Name, File Type, File size”
EXAMPLE	Check the files and directories whose path is “Local/”: *MMEMory:CATalog? “Local”* Check the files with “.arb” or “.ARB” postfix whose path is “Local/”: *MMEMory:CATalog:DATA:ARBitrary? “”* Check the files with “.xml” or ”.XML” postfix whose path is “Local/”: *MMEMory:CATalog:STATe:XMLanguage? “”*

3.41.5 MMEMory:COPY

DESCRIPTION	This command copies a file or a directory .
COMMAND SYNTAX	MMEMory:COPY <parameter> <parameter>:= ”The path of the source”, “The path that the file is about to pasted to”.
EXAMPLE	Copy the file whose path is ”Local/test/1000pts.bin” and paste to “Local/1000pts.bin” *MMEMory:COPY “1000pts.bin”,”1000pts.bin”* Copy the directory whose path is ”Local/src” and paste to “Local/copy/” *MMEMory:COPY “src”, ”copy”*

3.41.6 MMEMory:MOVE

DESCRIPTION	This command movesthe file or the directory to a new location.
COMMAND SYNTAX	MMEMory:MOVE<parameter> <parameter>:= ”The path of the source”, “The path of the source that is about to move to”.
QUERY SYNTAX	
RESPONSE FORMAT	
EXAMPLE	Move the file whose path is ”Local/test/1000pts.bin” to “Local/1000pts.bin” *MMEMory:MOVE “test/1000pts.bin”,”1000pts.bin”* Move the directory whose path is ”Local/src” to “Local/copy/paste” *MMEMory:MOVE “src”, ”copy/”*

3.41.7 MMEMory:SAVE:XML

DESCRIPTION	This command saves an xml configuration file to the default path or specified path
COMMAND SYNTAX	MMEMory:SAVE:XML<parameter> <parameter>:= {save path, including file name and suffix}.
QUERY SYNTAX	
RESPONSE FORMAT	
EXAMPLE	Save the test.xml file to the local *MMEMory:SAVE:XML "Local/test.xml" or* *MMEMory:SAVE:XML "test.xml"*

3.41.8 MMEMory:LOAD:XML

DESCRIPTION	Load an xml configuration file from the default path or the specified path
COMMAND SYNTAX	MMEMory:LOAD:XML<parameter> <parameter>:= {path, including file name and suffix}.
QUERY SYNTAX	
RESPONSE FORMAT	
EXAMPLE	Load the test.xml file locally *MMEMory:SAVE:XML "Local/test.xml" or* *MMEMory:SAVE:XML "test.xml"* Load the test.xml file from the USB flash drive *MMEMory:SAVE:XML "U-disk0/test.xml"*

3.41.9 MMEMory:TRANsfer

DESCRIPTION	This command can send customized data to the specified bin file in the specified path
COMMAND SYNTAX	MMEMory:TRANsfer < parameter >,#{data} <parameter>:= {path, including file name and suffix}. {data}:= Length of data length+data length+binary data
QUERY SYNTAX	
RESPONSE FORMAT	
EXAMPLE	Send the data with the length of 1 and the length of 4 to the local wave1.bin *MMEMory:TRANsfer "Local/wave1.bin",#14ABCD*

4 Waveform format

This chapter gives a description of the waveform file format used by the signal source. Through these instructions, you can learn how to customize and edit the waveform file.

4.1 bin

The bin file content is binary, and the file content is the codeword value of each point (codeword range – 32768~32767). Manual editing is not supported. When the machine imports the file, it maintains the current amplitude, frequency and offset information, and directly converts each codeword value into voltage output.

4.2 csv/dat

The contents of the csv and dat files are text. The CSV file supported by SDG6000X is divided into header information and waveform data.

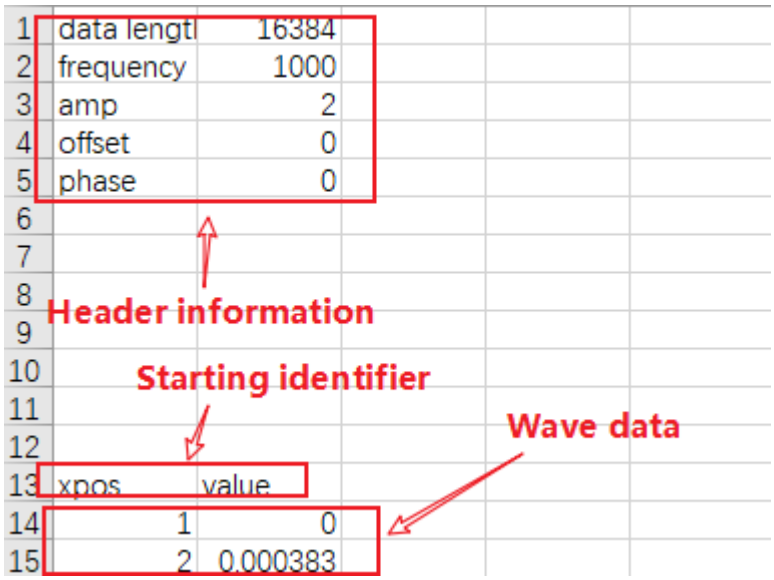
1. The content of the csv header information contains the following four items. There can be more information items, but the following four items must be included, and more items will be ignored. One line for each item, ending with a newline character

Header information	Describe
amp, value	Amplitude of waveform
offset, value	Offset of waveform
frequency, value	Frequency of waveform
data length, value	Length of waveform

2. Each group of segment data occupies one row. One of the following four strings can be used as the starting identifier (identifier occupies one line), which is placed before the wave data.

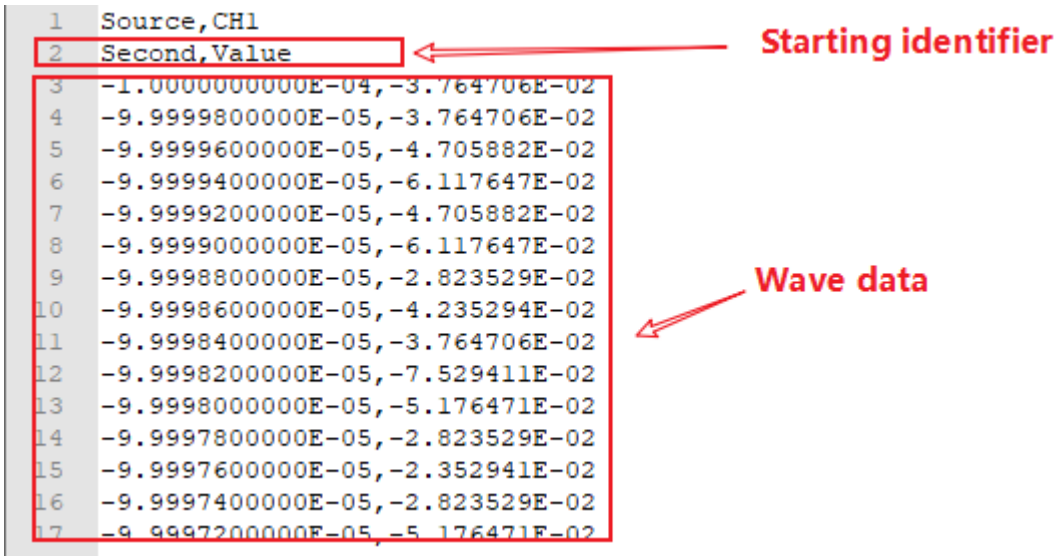
Starting identifier
Second, Volt
Xpos, value
Second, Volt
Second, Value

An example of csv format data is shown below:



The header information is removed from the csv file, and only the data is retained, which is dat format.

An example of dat format data is shown below:



5 Programming Examples

This chapter gives some examples for the programmer. In these examples, you can see how to use VISA or sockets, in combination with the commands described above to control the generator. By following these examples, you can develop many more applications.

5.1 Examples of Using VISA

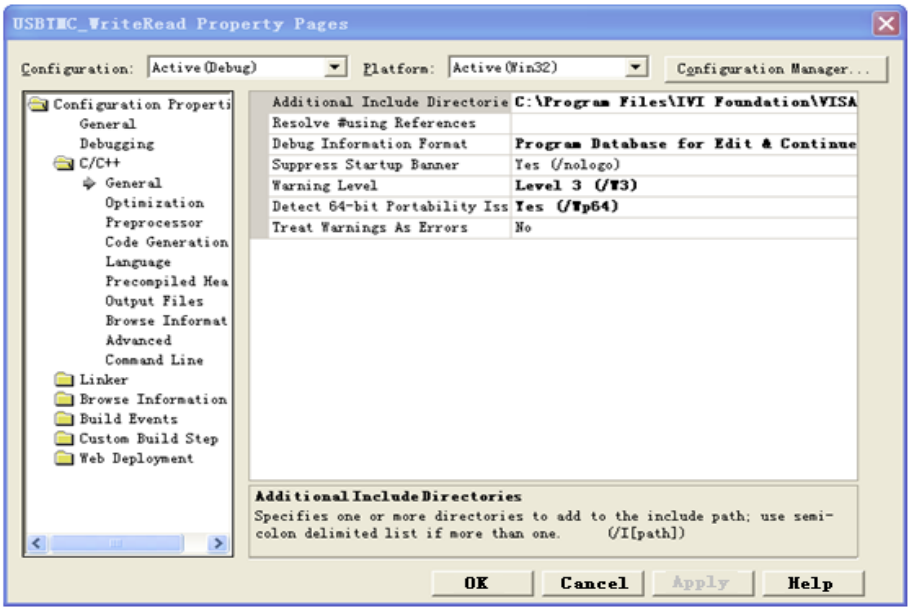
5.1.1 VC++ Example

Environment: Windows 7 32-bit, Visual Studio.
Description: Query the instrument information using "*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

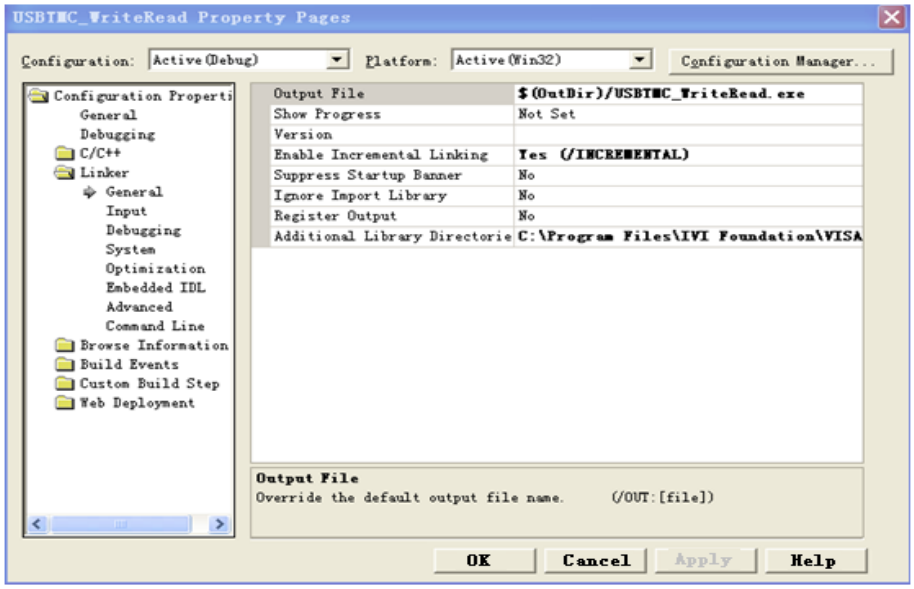
Steps:

1. Open Visual Studio and create a new VC++ win32 console project.

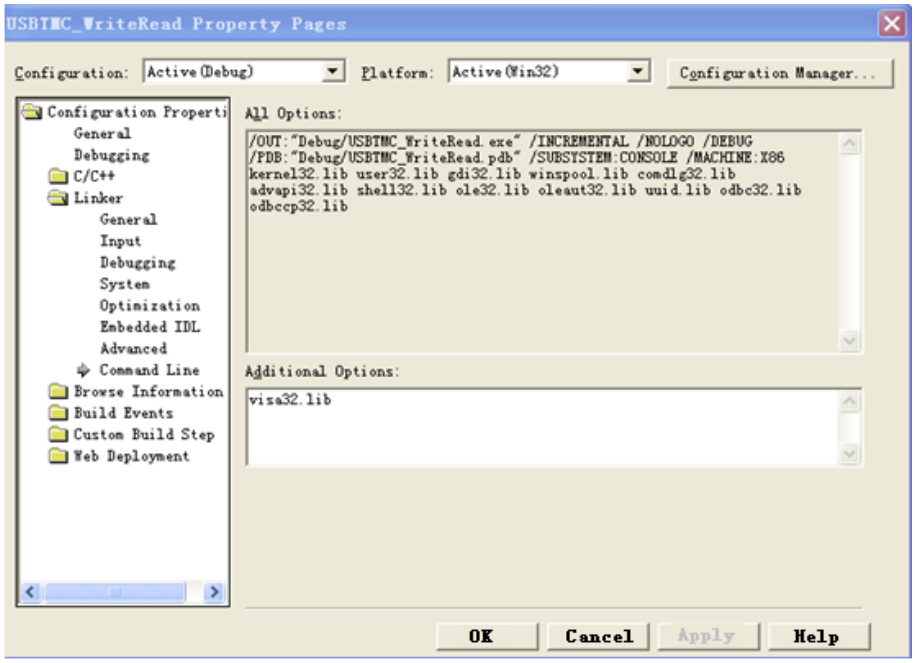
2. Set the project environment to use the NI-VISA lib, there are two ways to specify NI-VISA, static or automatic:
   a) Static:
   Find the files visa.h, visatype.h, and visa32.lib in the NI-VISA installation path, copy them to the root path of the VC++ project, and add them to the project. In the project name.cpp file, add the following two lines:
   ```
   #include "visa.h"
   #pragma comment (lib,"visa32.lib")
   ```
   b) Dynamic:
   In "project---properties---c/c++---General---Additional Include Directories" set the value to the NI-VISA installation path (e.g. C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the figure below:



In " project---properties---Linker---General---Additional Library Directories " set the value to the NI-VISA installation path (e.g. C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the figure below:



In "project---properties---Linker---Command Line---Additional" set the value to visa32.lib, as shown in the figure below:



Include visa.h file in the projectname.cpp file:

```
#include <visa.h>
```

3. Coding:

   a) USBTMC:

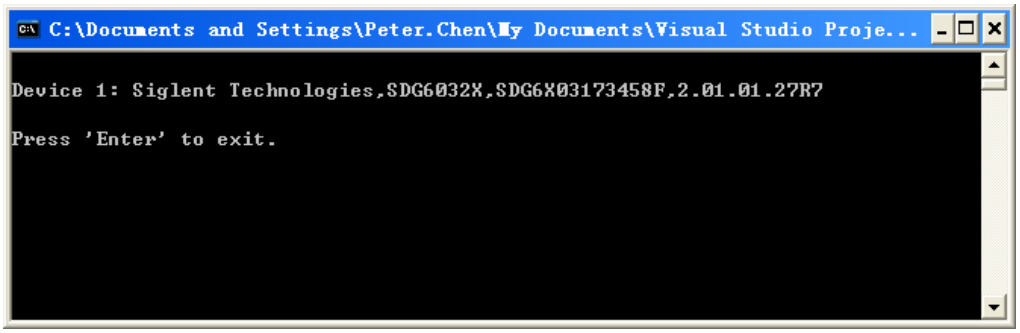
```
int Usbtmc_test()
{
    /* This code demonstrates sending synchronous read & write commands */
    /* to an USB Test & Measurement Class (USBTMC) instrument using      */
    /* NI-VISA                                                           */
    /* The example writes the "*IDN?\n" string to all the USBTMC         */
    /* devices connected to the system and attempts to read back         */
    /* results using the write and read functions.                       */
    /* The general flow of the code is   */
    /*      Open Resource Manager                */
    /*      Open VISA Session to an Instrument                           */
    /*      Write the Identification Query Using viPrintf                */
    /*      Try to Read a Response With viScanf                          */
    /*      Close the VISA Session               */
    /************************************************************/
    ViSession    defaultRM;
    ViSession    instr;
    ViUInt32     numInstrs;
    ViFindList   findList;
    ViStatus     status;
```

```
    char    instrResourceString[VI_FIND_BUFLEN];
    unsigned        char    buffer[100];
    int     i;
    /** First we must call viOpenDefaultRM to get the manager
    * handle.  We will store this handle in defaultRM.*/
    status=viOpenDefaultRM (&defaultRM);
    if (status<VI_SUCCESS)
    {
            printf ("Could not open a session to the VISA Resource Manager!\n");
            return  status;
    }
    /* Find all the USB TMC VISA resources in our system and store the  number of resources in the system in
numInstrs.                      */
    status = viFindRsrc (defaultRM, "USB?*INSTR", &findList, &numInstrs, instrResourceString);
    if (status<VI_SUCCESS)
    {
            printf ("An error occurred while finding resources.\nPress 'Enter' to continue.");
            fflush(stdin);
            getchar();
            viClose (defaultRM);
            return  status;
    }
    /** Now we will open VISA sessions to all USB TMC instruments.
    * We must use the handle from viOpenDefaultRM and we must
    * also use a string that indicates which instrument to open.  This
    * is called the instrument descriptor.  The format for this string
    * can be found in the function panel by right-clicking on the
    * descriptor parameter. After opening a session to the
    * device, we will get a handle to the instrument which we
    * will use in later VISA functions.  The AccessMode and Timeout
    * parameters in this function are reserved for future
    * functionality.  These two parameters are given the value VI_NULL.*/
    for (i=0; i<int(numInstrs); i++)
    {
            if (i> 0)
            {
                    viFindNext (findList, instrResourceString);
            }
            status = viOpen (defaultRM, instrResourceString, VI_NULL, VI_NULL, &instr);
            if (status<VI_SUCCESS)
            {
                    printf ("Cannot open a session to the device %d.\n", i+1);
```

```
                continue;
        }
        /* * At this point we now have a session open to the USB TMC instrument.
        * We will now use the viPrintf function to send the device the string "*IDN?\n",
        * asking for the device's identification.    */
        char * cmmand ="*IDN?\n";
        status = viPrintf    (instr, cmmand);
        if (status<VI_SUCCESS)
        {
                printf ("Error writing to the device %d.\n", i+1);
                status = viClose (instr);
                continue;
        }
        /** Now we will attempt to read back a response from the device to
        * the identification query that was sent.    We will use the viScanf
        * function to acquire the data.
        * After the data has been read the response is displayed.*/
        status =    viScanf(instr, "%t", buffer);
        if (status<VI_SUCCESS)
        {
                printf ("Error reading a response from the device %d.\n", i+1);
        }
        else
        {
                printf ("\nDevice %d: %s\n", i+1 , buffer);
        }
        status = viClose (instr);

    }
    /** Now we will close the session to the instrument using
    * viClose. This operation frees all system resources.                    */
    status = viClose (defaultRM);
    printf("Press 'Enter' to exit.");
    fflush(stdin);
    getchar();
    return 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    Usbtmc_test();
    return 0;
```

```
}
```

Run result:

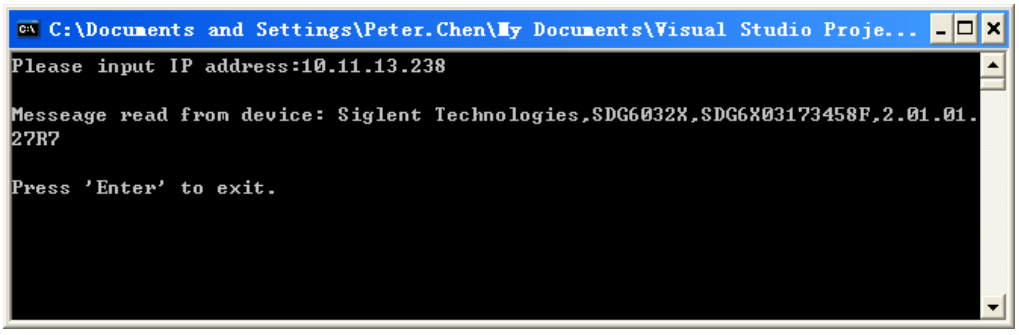


b) TCP/IP:

```
int TCP_IP_Test(char *pIP)
{
    char outputBuffer[VI_FIND_BUFLEN];
    ViSession defaultRM, instr;
    ViStatus status;
    /* First we will need to open the default resource manager. */
    status = viOpenDefaultRM (&defaultRM);
    if (status<VI_SUCCESS)
    {
            printf("Could not open a session to the VISA Resource Manager!\n");
    }
    /* Now we will open a session via TCP/IP device */
    char head[256] ="TCPIP0::";
    char tail[] ="::INSTR";
    strcat(head,pIP);
    strcat(head,tail);
    status = viOpen (defaultRM, head, VI_LOAD_CONFIG, VI_NULL, &instr);
    if (status<VI_SUCCESS)
    {
            printf ("An error occurred opening the session\n");
            viClose(defaultRM);
    }
    status = viPrintf(instr, "*idn?\n");
    status = viScanf(instr, "%t", outputBuffer);
    if (status<VI_SUCCESS)
    {
            printf ("viRead failed with error code: %x \n",status);
```

```
            viClose(defaultRM);
    }
    else
    {
            printf ("\nMesseage read from device: %*s\n", 0,outputBuffer);
    }
    status = viClose (instr);
    status = viClose (defaultRM);
    printf("Press 'Enter' to exit.");
    fflush(stdin);
    getchar();
    return 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    printf("Please input IP address:");
    char ip[256];
    fflush(stdin);
    gets(ip);
    TCP_IP_Test(ip);
    return 0;
}
```

Run result:



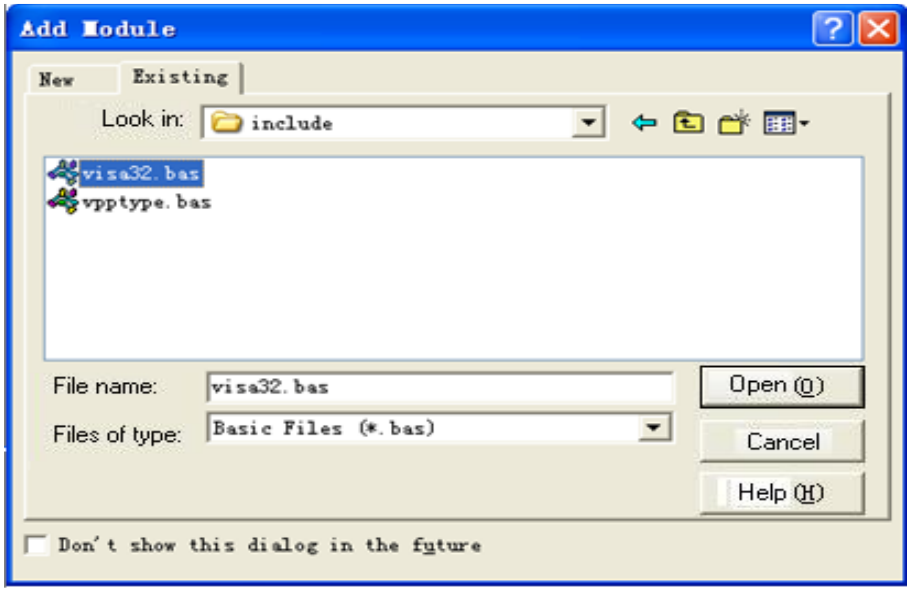
5.1.2 VB Example

Environment: Windows 7 32-bit, Microsoft Visual Basic 6.0

Description: Query the instrument information using the "*IDN?" command over NI-VISA, via USBTMC and TCP/IP separately.

Steps:

1. Open Visual Basic, and build a standard application program project.
2. Set the project environment to use the NI-VISA lib: Click the Existing tab of Project>>Add Existing Item, search the visa32.bas file in the "include" folder under the NI-VISA installation path and add the file, as shown in the figure below:



3. Coding:
   a) USBTMC:

```
Private Function Usbtmc_test() As Long
    ' This code demonstrates sending synchronous read & write commands
    ' to an USB Test & Measurement Class (USBTMC) instrument using
    ' NI-VISA
    ' The example writes the "*IDN?\n" string to all the USBTMC
    ' devices connected to the system and attempts to read back
    ' results using the write and read functions.
    ' The general flow of the code is
    '      Open Resource Manager
    '      Open VISA Session to an Instrument
    '      Write the Identification Query Using viWrite
    '      Try to Read a Response With viRead
    '      Close the VISA Session
    Const MAX_CNT = 200

    Dim defaultRM As Long
    Dim instrsesn As Long
    Dim numInstrs As Long
    Dim findList As Long
```

```
Dim retCount As Long
Dim status As Long
Dim instrResourceString As String * VI_FIND_BUFLEN
Dim Buffer As String * MAX_CNT
Dim I As Integer

‘‘ First we must call viOpenDefaultRM to get the manager
‘‘ handle. We will store this handle in defaultRM.
status = viOpenDefaultRM(defaultRM)
If (status < VI_SUCCESS) Then
    resultTxt.Text =“"Could not open a session to the VISA Resource Manager”"
    Usbtmc_test = status
    Exit Function
End If

‘‘ Find all the USB TMC VISA resources in our system and store the
‘‘ number of resources in the system in numInstrs.
status = viFindRsrc(defaultRM,“"USB?*INST”", findList, numInstrs, instrResourceString)
If (status < VI_SUCCESS) Then
    resultTxt.Text =“"An error occurred while finding resources”"
    viClose(defaultRM)
    Usbtmc_test = status
    Exit Function
End If

‘‘ Now we will open VISA sessions to all USB TMC instruments.
‘‘ We must use the handle from viOpenDefaultRM and we must
‘‘ also use a string that indicates which instrument to open.  This
‘‘ is called the instrument descriptor.  The format for this string
‘‘ can be found in the function panel by right-clicking on the
‘‘ descriptor parameter. After opening a session to the
‘‘ device, we will get a handle to the instrument which we
‘‘ will use in later VISA functions.  The AccessMode and Timeout
‘‘ parameters in this function are reserved for future
‘‘ functionality.  These two parameters are given the value VI_NULL.
For i = 0 To numInstrs
    If (i > 0) Then
        status = viFindNext(findList, instrResourceString)
    End If
    status = viOpen(defaultRM, instrResourceString, VI_NULL, VI_NULL, instrsesn)
    If (status < VI_SUCCESS) Then
        resultTxt.Text =“"Cannot open a session to the device“" + CStr(i + 1)
```

```
            GoTo NextFind
        End If

    '' At this point we now have a session open to the USB TMC instrument.
    '' We will now use the viWrite function to send the device the string""*IDN"",
    '' asking for the devic"s identification.
        status = viWrite(instrsesn,""*IDN"", 5, retCount)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =""Error writing to the device""
            status = viClose(instrsesn)
            GoTo NextFind
        End If
    '' Now we will attempt to read back a response from the device to
    '' the identification query that was sent.    We will use the viRead
    '' function to acquire the data.
    '' After the data has been read the response is displayed.
        status = viRead(instrsesn, Buffer, MAX_CNT, retCount)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =""Error reading a response from the device"" + CStr(i + 1)
        Else
            resultTxt.Text =""Read from device:"" + CStr(i + 1) +""""" + Buffer
        End If
        status = viClose(instrsesn)
    Next i

'' Now we will close the session to the instrument using
'' viClose. This operation frees all system resources.
    status = viClose(defaultRM)
    Usbtmc_test = 0
End Function
```

b) TCP/IP:

```
Private Function TCP_IP_Test(ByVal ip As String) As Long
    Dim outputBuffer As String * VI_FIND_BUFLEN
    Dim defaultRM As Long
    Dim instrsesn As Long
    Dim status As Long
    Dim count As Long

'' First we will need to open the default resource manager.
    status = viOpenDefaultRM(defaultRM)
    If (status < VI_SUCCESS) Then
```

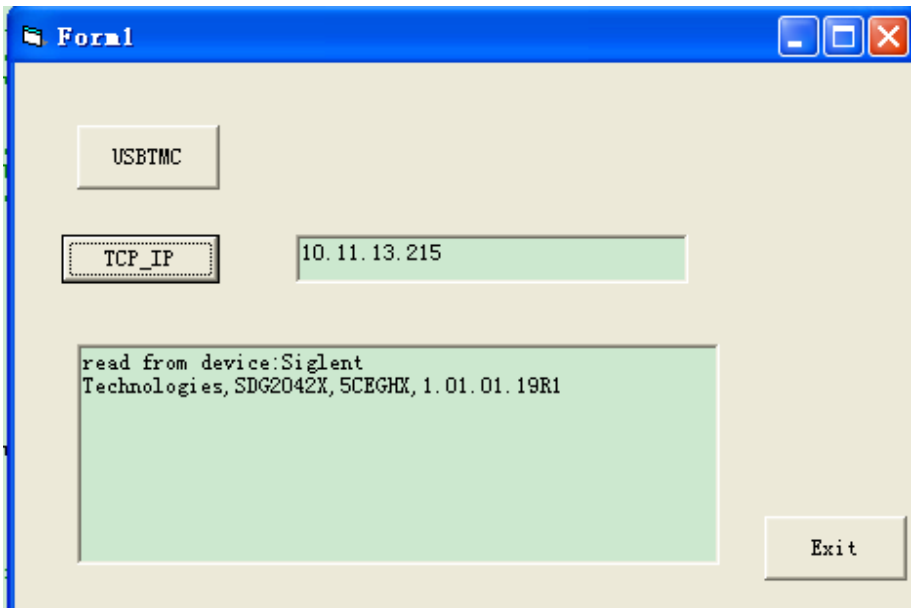
```
            resultTxt.Text =""Could not open a session to the VISA Resource Manager""
            TCP_IP_Test = status
            Exit Function
        End If

    '' Now we will open a session via TCP/IP device
        status = viOpen(defaultRM,""TCPIP0:"" + ip +""::INST"", VI_LOAD_CONFIG, VI_NULL, instrsesn)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =""An error occurred opening the sessio""
            viClose(defaultRM)
            TCP_IP_Test = status
            Exit Function
        End If

        status = viWrite(instrsesn,""*IDN"", 5, count)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =""Error writing to the device""
        End If
        status = viRead(instrsesn, outputBuffer, VI_FIND_BUFLEN, count)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =""Error reading a response from the device"" + CStr(i + 1)
        Else
            resultTxt.Text =""read from device"" + outputBuffer
        End If
        status = viClose(instrsesn)
        status = viClose(defaultRM)
        TCP_IP_Test = 0
    End Function
```

c) Button control code:

```
    Private Sub exitBtn_Click()
        End
    End Sub
    Private Sub tcpipBtn_Click()
        Dim stat As Long
        stat = TCP_IP_Test(ipTxt.Text)
        If (stat < VI_SUCCESS) Then
            resultTxt.Text = Hex(stat)
        End If
    End Sub
    Private Sub usbBtn_Click()
        Dim stat As Long
```

```
        stat = Usbtmc_test
        If (stat < VI_SUCCESS) Then
            resultTxt.Text = Hex(stat)
        End If
    End Sub
```

Run result:



5.1.3 MATLAB Example

Environment: Windows 7 32-bit, MATLAB R2013a

Description: Query the instrument information using the "*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

Steps:

1. Open MATLAB, and modify the current directory. In this demo, the current directory is modified to "D:\USBTMC_TCPIP_Demo".

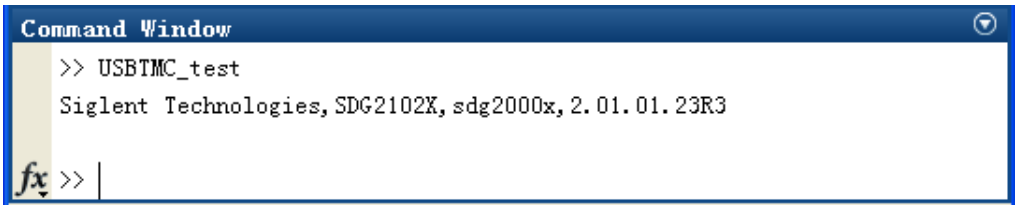
2. Click File>>New>>Script in the Matlab interface to create an empty M file.

3. Coding:
   a) USBTMC:

```
function USBTMC_test()
% This code demonstrates sending synchronous read & write commands
% to an USB Test & Measurement Class (USBTMC) instrument using
```

```
% NI-VISA

%Create a VISA-USB object connected to a USB instrument
vu = visa('ni','USB0::0xF4ED::0xEE3A::sdg6000x::INSTR');

%Open the VISA object created
fopen(vu);

%Send the string "*IDN?",asking for the device's identification.
fprintf(vu,'*IDN?');

%Request the data
outputbuffer = fscanf(vu);
disp(outputbuffer);

%Close the VISA object
fclose(vu);
delete(vu);
clear vu;

end
```

Run result:



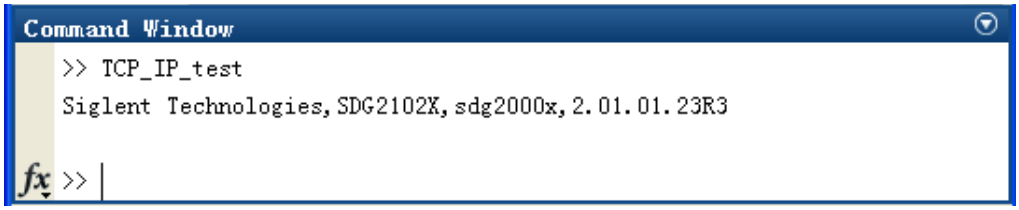
b) TCP/IP:

Write a function TCP_IP_Test:

```
function TCP_IP_test()
% This code demonstrates sending synchronous read & write commands
% to a TCP/IP instrument using NI-VISA

%Create a VISA-TCPIP object connected to an instrument
%configured with IP address.
vt = visa('ni',['TCPIP0::','10.11.13.32','::INSTR']);

%Open the VISA object created
fopen(vt);
```

```
%Send the string "*IDN?",asking for the device's identification.
fprintf(vt,'*IDN?');

%Request the data
outputbuffer = fscanf(vt);
disp(outputbuffer);

%Close the VISA object
fclose(vt);
delete(vt);
clear vt;

end
```

Run result:



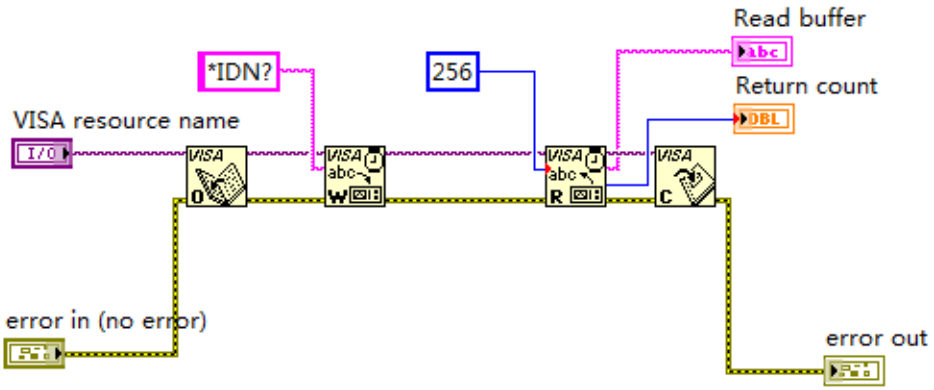
5.1.4 LabVIEW Example

Environment: Windows 7 32-bit, LabVIEW 2011

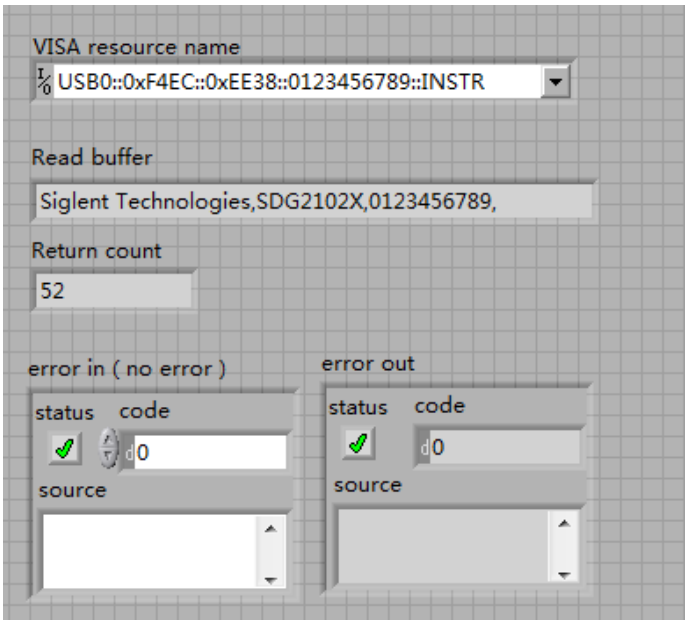
Description: Query the instrument information using the “*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

Steps:

1. Open LabVIEW, and create a VI file.
2. Add controls. Right-click in the **Front Panel** interface, select and add **VISA resource name**, error in, error out and some indicators from the Controls column.
3. Open the **Block Diagram** interface. Right-click on the **VISA resource name**, select and add the following functions from VISA Palette from the pop-up menu: **VISA Write, VISA Read, VISA Open,** and **VISA Close.**
4. The connection is as shown in the figure below:



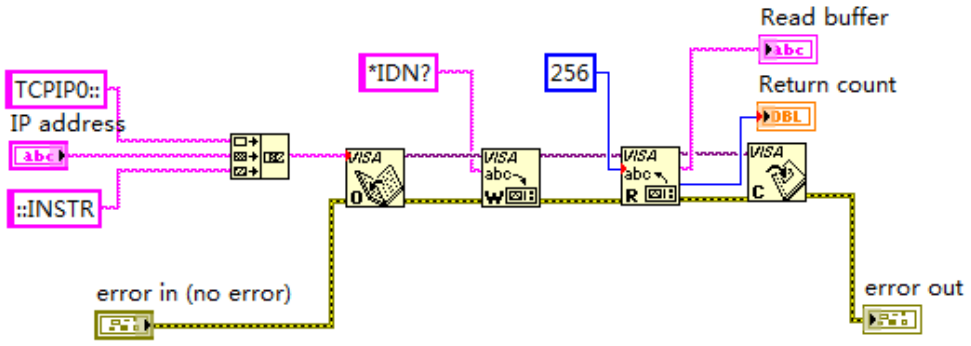
5. Select the device resource from the VISA Resource Name list box and run the program.



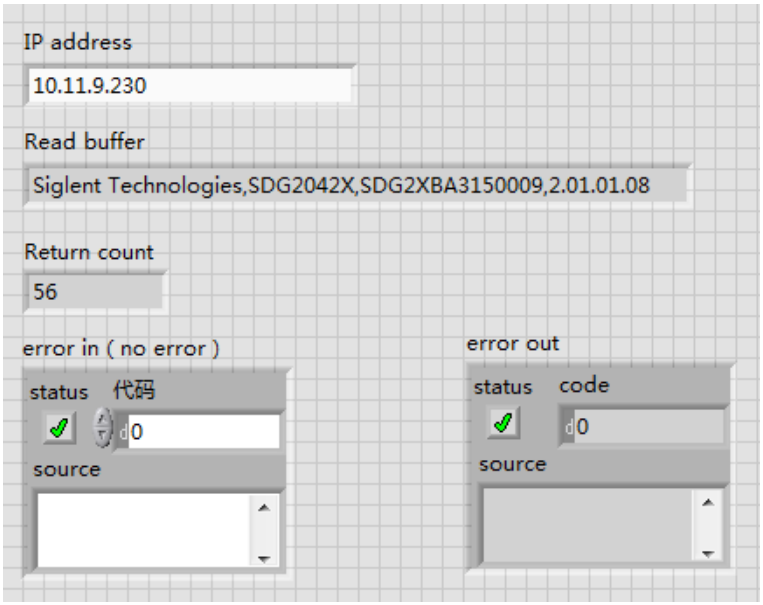
In this example, the VI opens a VISA session to a USBTMC device, writes a "*IDN?" command to the device, and reads back the response. After all communication is complete, the VI closes the VISA session.

6. Communicating with the device via TCP/IP is similar to USBTMC. But you need to change VISA Write and VISA Read Function to Synchronous I/O. The LabVIEW default is asynchronous I/O. Right-click the node and select Synchronous I/O Mod>>Synchronous from the shortcut menu to write or read data synchronously.

7. The connection is as shown in the figure below:



8. Input the IP address and run the program.



5.1.5 Python3 Example

Environment: Python3.6.5， PyVISA 1.9

Description: The number of waveform points created in the example represents the code word of each waveform point, which is used to determine the level of waveform points (the code word range of each waveform point is -32767~32767).

```
#!/usr/bin/env python3.6.5
# -*- coding: utf-8 -*-

import pyvisa as visa
import struct

#USB resource of Device
sdg=visa.ResourceManager().open_resource("USB0::0xF4EC::0x1105::SDG6000X002::INSTR")
data = [32767, 32767, 32767, 32767, 32767, 32767, 16384,16384,32767, 32767, 32767, 32767, 32767,
32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -16384,-16384,-32767, -32767,
-32767, -32767, -32767, -32767]

def int_to_two_byte_small_endian(n):
    return struct.pack('<h', n)    # '<h'

final_byte_data = b''.join(int_to_two_byte_small_endian(num) for num in data)
print('write bytes:', len(final_byte_data))
print(final_byte_data)
cmd = bytes('C1:WVDT WVNM,wave1,FRQ,2500,AMP,2.5,OFST,0.2,WAVEDATA,', 'utf-8') +
final_byte_data
sdg.write_raw(cmd)

f = open(r"C:\Users\Administrator\Desktop\wave1.bin", "rb")
data1 = f.read()
print ('write bytes:',len(data1))
cmd = bytes('C1:WVDT WVNM,wave1,FRQ,2500,AMP,2.5,OFST,0.2,WAVEDATA,', 'utf-8') + data1
sdg.write_raw(cmd)
f.close()
```

5.2 Examples of Using Sockets

5.2.1 Python Example

Python has a low-level networking module that provides access to the socket interface. Python scripts can be written for sockets to do a variety of tests and measurement tasks.

Environment: Windows 7 32-bit, Python v2.7.5

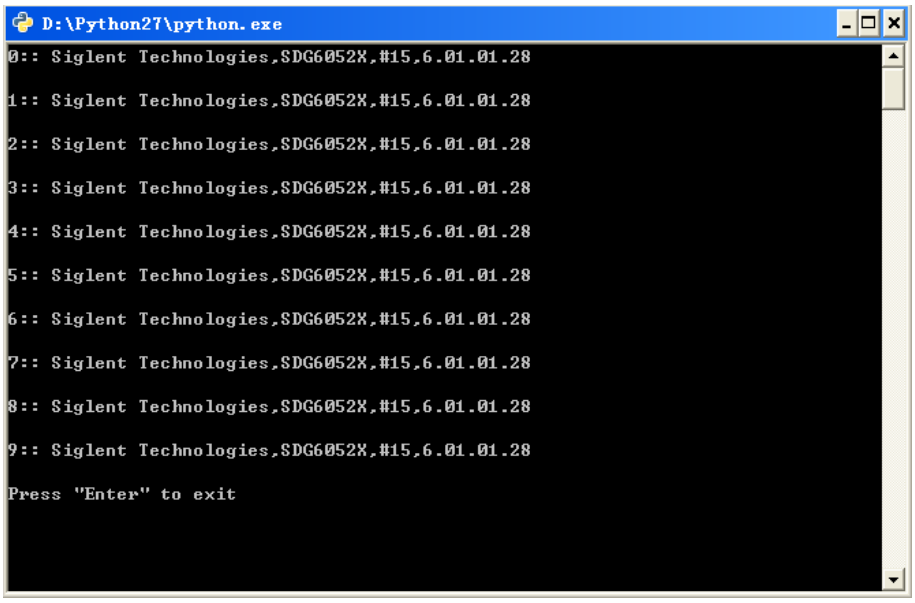
Description: Open a socket, send a query, and repeat this loop 10 times, finally close the socket. Note that SCPI command strings must be terminated with a "\n" (new line) character in programming.

Below is the code of the script:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
#------------------------------------------------------------------------------
# The short script is an example that opens a socket, sends a query,
# print the return message and closes the socket.
#------------------------------------------------------------------------------

import socket      # for sockets
import sys    # for exit
import time # for sleep

#------------------------------------------------------------------------------

remote_ip = "10.11.13.40"    # should match the instrument's IP address
port = 5025 # the port number of the instrument service
count = 0

def SocketConnect():
    try:
        #create an AF_INET, STREAM socket (TCP)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error:
        print ('Failed to create socket.')
        sys.exit();
    try:
        #Connect to remote server
        s.connect((remote_ip , port))
    except socket.error:
        print ('failed to connect to ip ' + remote_ip)
    return s
```

```
def SocketQuery(Sock, cmd):
    try :
        #Send cmd string
        Sock.sendall(cmd)
        time.sleep(1)
    except socket.error:
        #Send failed
        print ('Send failed')
        sys.exit()
    reply = Sock.recv(4096)
    return reply

def SocketClose(Sock):
    #close the socket
    Sock.close()
    time.sleep(.300)

def main():
    global remote_ip
    global port
    global count

    # Body: send the SCPI commands “*IDN?” 10 times and print the return message
    s = SocketConnect()
    for i in range(10):
        qStr = SocketQuery(s, b'*IDN?\n')
        print (str(count) + ":: " + str(qStr))
        count = count + 1
    SocketClose(s)
    input('Press "Enter" to exit')

if __name__ == '__main__':
    proc = main()
```

Run result:


D:\Python27\python.exe
0:: Siglent Technologies,SDG6052X,#15,6.01.01.28
1:: Siglent Technologies,SDG6052X,#15,6.01.01.28
2:: Siglent Technologies,SDG6052X,#15,6.01.01.28
3:: Siglent Technologies,SDG6052X,#15,6.01.01.28
4:: Siglent Technologies,SDG6052X,#15,6.01.01.28
5:: Siglent Technologies,SDG6052X,#15,6.01.01.28
6:: Siglent Technologies,SDG6052X,#15,6.01.01.28
7:: Siglent Technologies,SDG6052X,#15,6.01.01.28
8:: Siglent Technologies,SDG6052X,#15,6.01.01.28
9:: Siglent Technologies,SDG6052X,#15,6.01.01.28
Press "Enter" to exit

6 Index

O

P

R

S

W

V

About SIGLENT

SIGLENT is an international high-tech company, concentrating on R&D, sales, production and services of electronic test & measurement instruments.

SIGLENT first began developing digital oscilloscopes independently in 2002. After more than a decade of continuous development, SIGLENT has extended its product line to include digital oscilloscopes, isolated handheld oscilloscopes, function/arbitrary waveform generators, RF/MW signal generators, spectrum analyzers, vector network analyzers, digital multimeters, DC power supplies, electronic loads and other general purpose test instrumentation. Since its first oscilloscope was launched in 2005, SIGLENT has become the fastest growing manufacturer of digital oscilloscopes. We firmly believe that today SIGLENT is the best value in electronic test & measurement.

Headquarters:

SIGLENT Technologies Co., Ltd
Add: Bldg No.4 & No.5, Antongda Industrial Zone,
3rd Liuxian Road, Bao'an District,
Shenzhen, 518101, China
Tel: + 86 755 3688 7876
Fax: + 86 755 3359 1582
Email: sales@siglent.com
Website: int.siglent.com

North America:

SIGLENT Technologies NA, Inc
Add: 6557 Cochran Rd Solon, Ohio 44139
Tel: 440-398-5800
Toll Free:877-515-5551
Fax: 440-399-1211
Email: support@siglentna.com
Website: www.siglentna.com

Europe:

SIGLENT Technologies Germany GmbH
Add: Staetzlinger Str. 70
86165 Augsburg, Germany
Tel: +49(0)-821-666 0 111 0
Fax: +49(0)-821-666 0 111 22
Email: info-eu@siglent.com
Website: www.siglenteu.com

Malaysia:

SIGLENT Technologies (M) Sdn.Bhd
Add: NO.6 Lorong Jelawat 4
Kawasan Perusahaan Seberang Jaya
13700, Perai Pulau Pinang
Tel: 006-04-3998964
Email: sales@siglent.com
Website: int.siglent.com

Follow us on
Facebook: SiglentTech