SDG3000X
Arbitrary Waveform Generator

Programming Guide

EN01A

SIGLENT TECHNOLOGIES CO.,LTD

Contents

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

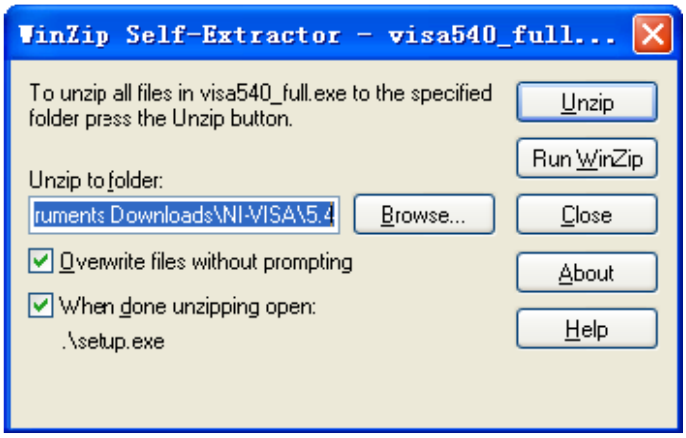
Before programming, please make sure that you have properly installed the latest version of National Instruments NI-VISA Software.

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):

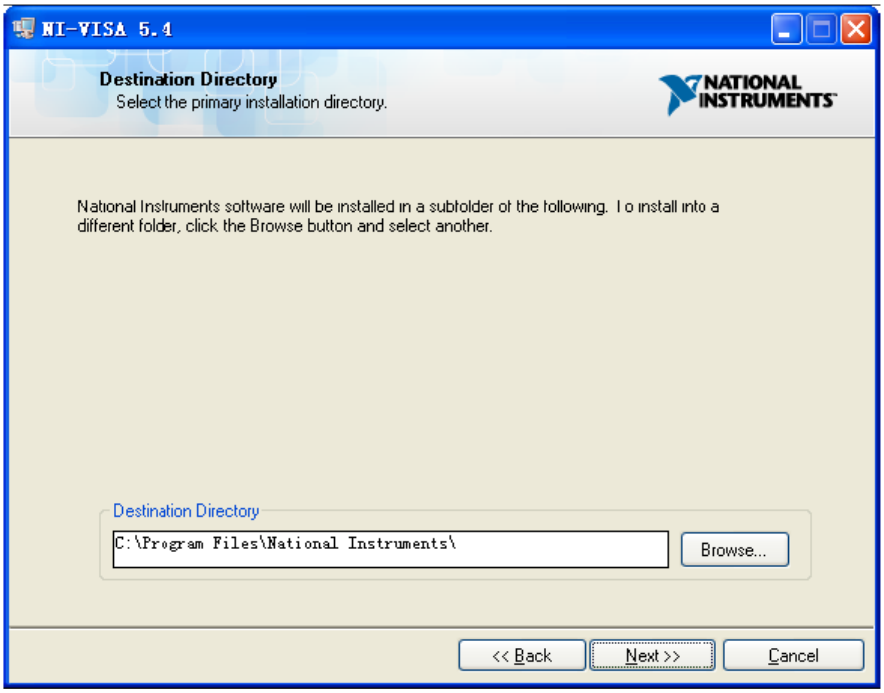
a. Download the appropriate version of NI-VISA (the Run-time engine is recommanded)

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

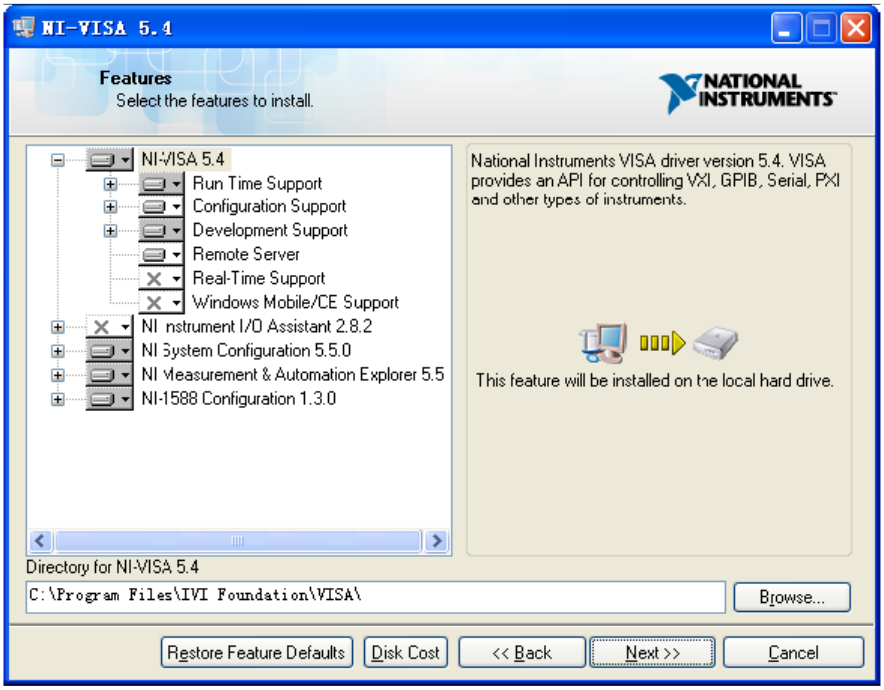


c. Click Unzip, the install process will launch after unzipping files. If your computer needs to install the .NET Framework 4, it may auto-start.

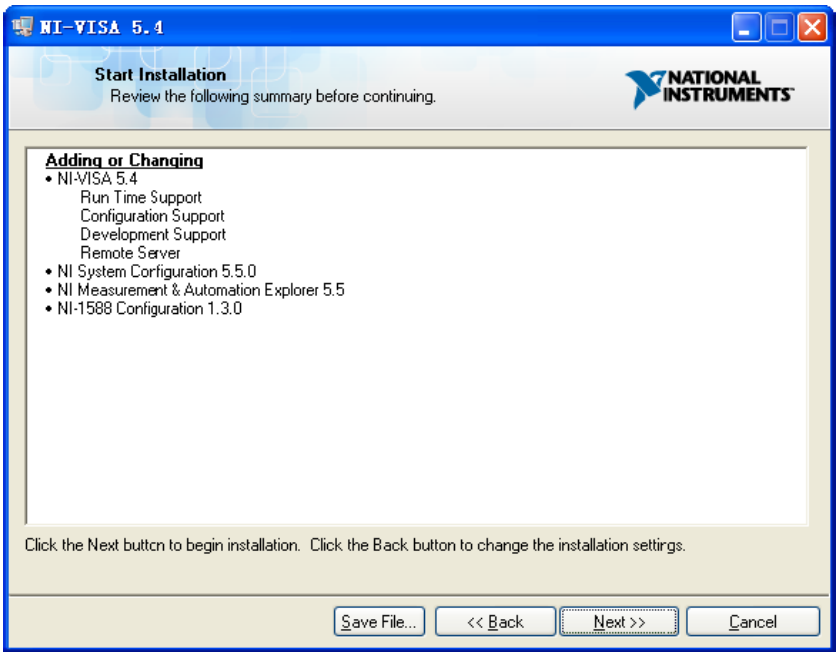
d. The NI-VISA install dialog is shown above. Click Next to start the installation process.



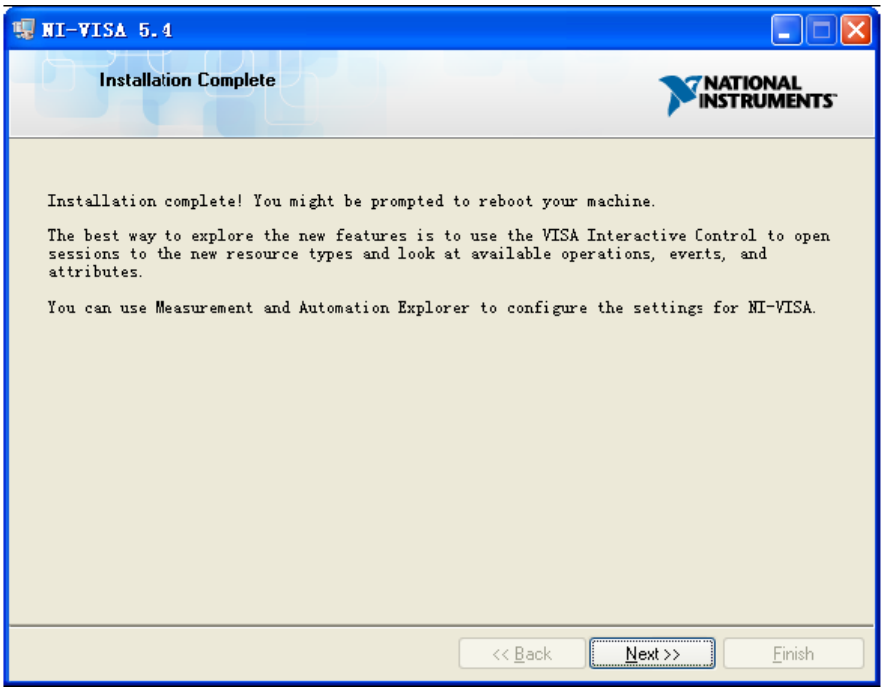
e. Set the install path, the default path is “C:\Program Files\National Instruments\”, you can change it if you prefer. Click Next, dialog as shown above.



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.

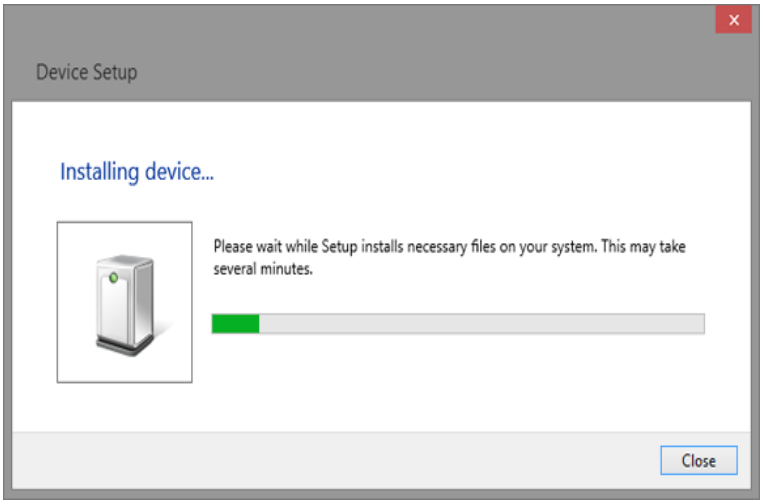


h. Now the installation is complete. Reboot your PC.

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".

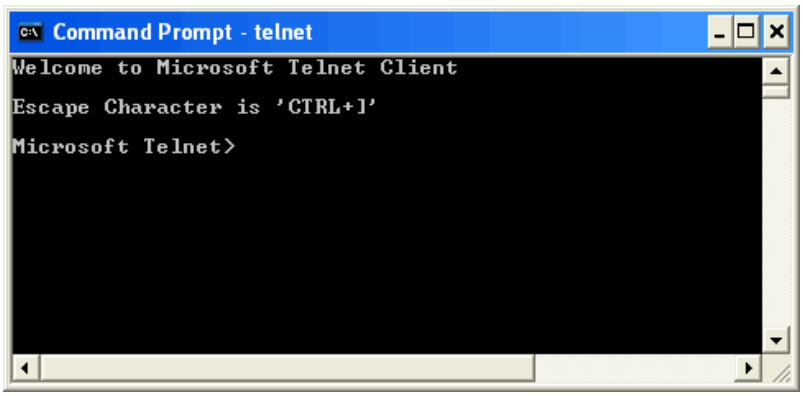
1.2.2 Using SCPI via NI-MAX

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.

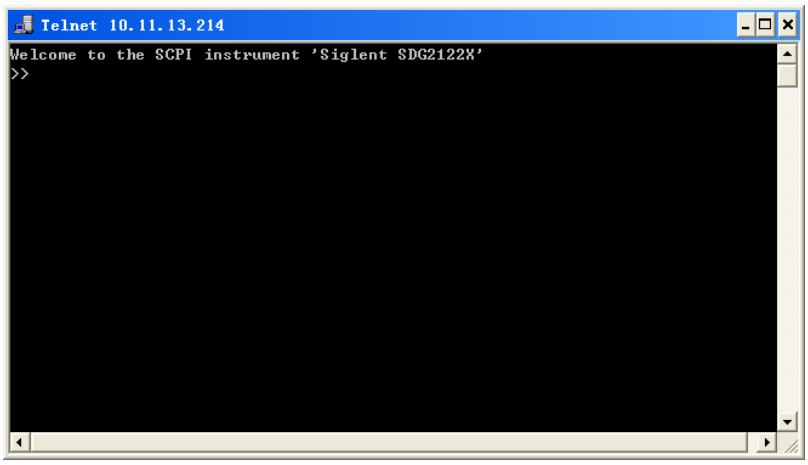
1.2.3 Using SCPI over Telnet

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:

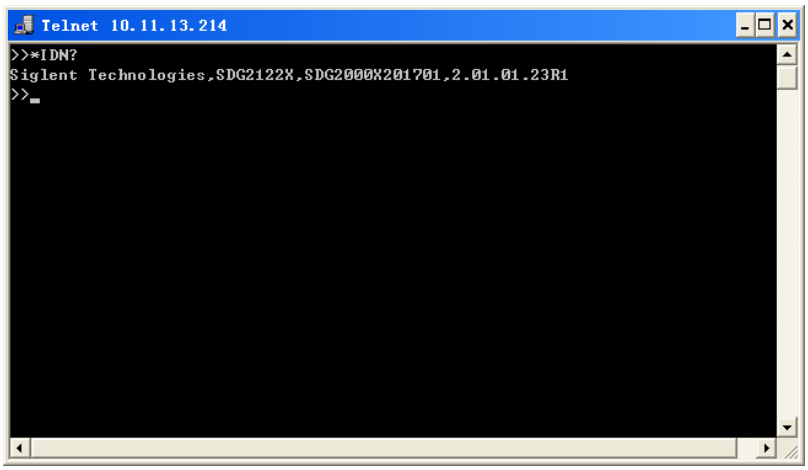
1. On your PC, click Start > All Programs > Accessories > Command Prompt.
2. At the command prompt, type in *telnet*.
3. Press the Enter key. The Telnet display screen will be displayed.



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.

NOTE: The TELNET client may not be enabled by default in Windows, You need to manually find the "Telnet Client" option in the "Turn Windows Features On or Off" option, click the check box and confirm.

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 "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 SDG3000X 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
OUTP	OUTPUT	SIGNAL	Sets or gets the output state.
BSWV	BASIC_WAVE	SIGNAL	Sets or gets the basic wave parameters.
PRBS	PRBS	SIGNAL	Sets or gets PRBS polynomial 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.
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	SYSTEM	Sets or gets the state of the clock source.
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.

Short	Long Form	Subsystem	What Command/Query does
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
IQ:CENT	IQ:CENTerfreq	SIGNAL	Sets the I/Q modulator center frequency.
IQ:SAMP	IQ:SAMPlerate	SIGNAL	Sets the I/Q sample rate.
IQ:SYMB	IQ:SYMBolrate	SIGNAL	Sets the I/Q symbol rate.
IQ:AMPL	IQ:AMPLitude	SIGNAL	Sets the I/Q amplitude.
IQ:IQAD:GAIN	IQ:IQADjustment:GAIN	SIGNAL	Adjusts the ratio of I to Q while preserving the composite.
IQ:IQAD:IOFF	IQ:IQADjustment:IOFFset	SIGNAL	Adjusts the I channel offset value.
IQ:IQAD:QOFF	IQ:IQADjustment:QOFFset	SIGNAL	Adjusts the I channel offset value.
IQ:IQAD:QSK	IQ:IQADjustment:QSKew	SIGNAL	Adjusts the phase angle between the I and Q vectors by increasing or decreasing the Q phase angle.
IQ:TRIG:SOUR	IQ:TRIGger:SOURce	SIGNAL	Sets the I/Q trigger source.
IQ:WAVE:BUIL	IQ:WAVEload:BUILtin	SIGNAL	Selects the I/Q wave from the built-in wave list.
IQ:WAVE:USER	IQ:WAVEload:USERstored	SIGNAL	Select the I/Q wave from user stored waveforms.
:IQ: FrequencySampling	:IQ: FrequencySampling	SIGNAL	Sets the I/Q Frequency sampling rate.

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,SDG3202X,SDG3000XABC002,1.1.1.5R3* **NOTE:** It may differ from each 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 impact on device operation. Because the instrument starts parsing the setting or query command after processing the previous setting or query statement. *OPC? The query command always returns ASCII code 1. Because the device does not respond to the query command until the previous command has been completely executed.
COMMAND SYNTAX	*OPC
QUERY SYNTAX	*OPC?
RESPONSE FORMAT	1

3.1.3 *RST

DESCRIPTION	The *RST command initiates an instrument reset and recalls default settings.
COMMAND SYNTAX	*RST
EXAMPLE	This example resets the signal generator to its default settings: **RST*

3.2 Channel output command

DESCRIPTION	Enables and disables the output of the [OUTPUT] connector on the channel front panel. The query results return "ON", "OFF", LOAD and PLRT parameters.
COMMAND SYNTAX	<channel>:OUTPut ON|OFF,LOAD,<load>,PLRT,<polarity> <channel>:={C1, C2} <load>:={50, HZ} <polarity>:={NOR, INVT}, NOR stands for normal and INVT stands for inverted phase.
QUERY SYNTAX	<channel>:OUTPut?
RESPONSE FORMAT	<channel>:OUTP ON|OFF,LOAD,<load>,PLRT,<polarity>
EXAMPLE	Open CH1: *C1:OUTP ON* Read CH1 output status: *C1:OUTP?* Return: *C1:OUTP ON,LOAD,HZ,PLRT,NOR* Set the load of CH1 to 50Ω: *C1:OUTP LOAD,50* Set the load of CH1 to high impedance: *C1:OUTP LOAD,HZ* Set the polarity of CH1 to normal: *C1:OUTP PLRT,NOR*

DESCRIPTION	Set the output status of two channels at the same time.
COMMAND SYNTAX	OUT_BOTHCH <STATE> <STATE>:={ON, OFF}
QUERY SYNTAX	
RESPONSE FORMAT	
EXAMPLE	Two channels open output: *OUT_BOTHCH ON*

3.3 Channel mode selection command

DESCRIPTION	This command is used to set or read the current mode of a channel.
COMMAND SYNTAX	<channel>:CPARam MODE,< parameter > <channel>:={C1, C2} < parameter >:={AFG, AWG, IQ}
QUERY SYNTAX	<channel>:CPARam?
RESPONSE FORMAT	MODE,< parameter >
EXAMPLE	Set the current output mode of channel 1 to AFG: *C1:CPARam MODE,AFG* Query the output mode of the current channel: *C1:CPARam?* Return: *MODE,AFG*

3.4 Output setting command

3.4.1 Noise superposition command

DESCRIPTION	Turn on noise overlay 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(dB) > <channel>:={C1, C2} Please refer to the data sheet for the signal-to-noise ratio value.
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(dB) >
EXAMPLE	Turn on channel one noise superposition and set the signal-to-noise ratio to 120: *C1:NOISE_ADD STATE,ON,RATIO,120*

3.4.2 Digital filter command

DESCRIPTION	This command is used to set the switching and cutoff frequencies of the digital filter.
COMMAND SYNTAX	<channel>:FILTer < parameter >,< value > < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:={ON, OFF}, Digital filter switch status.
COFF_FRQ	<frequency>	:= The unit is Hz and is used to set the cutoff frequency of the digital filter.

QUERY SYNTAX	<channel>:FILTer?
RESPONSE FORMAT	<channel>:FILT < parameter >,< value >
EXAMPLE	Set the digital filter of CH1 to turn on: *C1:FILT STATE,ON*

	Set the digital filter cutoff frequency of CH1 to 200MHz: *C1:FILT COFF_FRQ,200000000* Query the digital filter information of CH1: *C1:FILT?* Return: *STATE,OFF,COFF_FRQ,200000000Hz*

3.4.3 Polarity flip command

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

3.4.4 Amplitude limit command

DESCRIPTION	This command is used to set or get the amplitude limit value.
COMMAND SYNTAX	<channel>:BSWV MAX_OUTPUT_AMP,< value> <channel>:={C1, C2} <value>:={ The value of the relevant parameter}
QUERY SYNTAX	<channel>:BSWV?
RESPONSE FORMAT	<channel>:BSWV < parameter > < parameter >:={ All parameters of the current waveform}
EXAMPLE	Set CH1 amplitude limit to 20V: *C1:BSWV MAX_OUTPUT_AMP,20*

3.4.5 Overvoltage protection command

DESCRIPTION	This command is used to set or get the overvoltage protection switch status.
COMMAND SYNTAX	<channel>:VOLTPRT < state > <channel>:={C1, C2} <state>:={ON, OFF}
QUERY SYNTAX	<channel>:VOLTPRT?
RESPONSE FORMAT	VOLTPRT < state >
EXAMPLE	Set CH1 overvoltage protection status to on: *C1:VOLTPRT ON* Read the current overvoltage protection switch status of CH1: *C1:VOLTPRT?* Return: *ON*

3.5 Multi-channel setting command

3.5.1 Phase mode setting command

DESCRIPTION	This parameter sets or gets the phase mode.
COMMAND SYNTAX	G1:MODE < parameter > < parameter >:={ PHASELOCKED, INDEPENDENT}
QUERY SYNTAX	G1:MODE?
RESPONSE FORMAT	MODE < parameter >
EXAMPLE	Set phase mode to phase independent mode: *G1:MODE INDEPENDENT*

3.5.2 Channel combine command

DESCRIPTION	This command sets or gets the waveform merge.
COMMAND SYNTAX	<channel>:CoMBiNe < state > < state >:={ ON, OFF}
QUERY SYNTAX	<channel>:CoMBiNe?
RESPONSE FORMAT	<channel>:CoMBiNe < state >
EXAMPLE	Turn on waveform merging of CH1: *C1:CoMBiNe ON* Query the waveform merging status of CH2: *C2:CMBN?* Return: *C2:CMBN OFF*

3.5.3 Eqphase command

DESCRIPTION	This command is used to set the phase synchronization of two channels.
COMMAND SYNTAX	G1:EQPHASE
RESPONSE FORMAT	EQPHASE < state >
EXAMPLE	Set the same phase: *G1:EQPHASE*

3.5.4 Channel tracking, coupling commands

DESCRIPTION	Set or get channel coupling parameters. Coupling values can only be set when tracking is turned off.
COMMAND SYNTAX	G1:COUPling < parameter >,<value> < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
TRACE	<track_enble>	:={ON, OFF}, channel tracking status.
FCOUP	<fcoup>	:={ON, OFF}, frequency coupling state.
FDEV	<frq_dev>	:=The frequency difference between the two channels. The unit is 'Hz'.
FRAT	<frat>	:=frequency ratio between two channels.
PCOUP	<pcoup>	:={ON, OFF}, phase coupling state.
PDEV	<pha_dev>	:=The phase difference between the two channels. The unit is 'degree'.
PRAT	<prat>	:=Phase ratio between two channels.
ACOUP	<acoup>	:={ON, OFF}, amplitude coupling state.
ARAT	<arat>	:=Amplitude ratio between two channels.
ADEV	<adev>	:=Amplitude deviation between two channels. The unit is 'Vpp'.

QUERY SYNTAX	G1:COUPling?
RESPONSE FORMAT	< parameter >:={ All coupling parameter values}
EXAMPLE	Set channel tracking status to open: *G1:COUP TRACE,ON* Set frequency deviation to 5Hz: *G1:COUP FDEV,5* Set the amplitude ratio to 2: *G1:COUP ARAT,2* Query coupling information: *G1:COUP?* Return: *TRACE,OFF,FCOUP,ON,PCOUP,ON,ACOUP,ON,FDEV,5HZ,PDEV,0,ARAT,2*

3.5.5 Channel copy command

DESCRIPTION	This command copies the parameters of one channel to another channel.
COMMAND SYNTAX	ParaCoPy < target channel >,< channel source > < target channel >:={C1, C2} < channel source >:={C1, C2}
EXAMPLE	Copy the parameters of channel 1 to channel 2: *PACP C2,C1*

3.6 AFG command

3.6.1 Basic waveform command

DESCRIPTION	Set or get basic wave parameters.
COMMAND SYNTAX	<channel>:BaSic_WaVe < parameter >,<value> <channel>:={C1, C2} < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
WVTP	<type>	:={Sine, Square, Ramp, Pulse, Noise, ARB, DC, PRBS}. If the command does not set a basic waveform type, WVPT defaults to the current waveform.
FRQ	<frequency>	:= frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This value is invalid when WVTP is noise, DC and multi-pulse.
PERI	<period>	:=Period. The unit is "s". The valid range of parameter values can be found in the data sheet. When WVTP is noise, DC and multi-pulse, this value is invalid.
AMP	<amplitude>	:=Amplitude. The unit is "Vpp". The valid range of parameter values can be found in the data sheet. When WVTP is noise and DC, this parameter is invalid.
OFST	<offset>	:=Offset. The unit is volt "V". The valid range of parameter values can be found in the data sheet. When WVTP is noise, this value has no effect.
SYM	<symmetry>	:= {0 to 100}. Symmetry of the triangle wave. The unit is "%". This parameter can be set only when WVTP is a triangle wave.
DUTY	<duty>	:= {0 to 100}. duty cycle. The unit is "%". The value of this parameter depends on the frequency. This parameter can only be set when WVTP is square wave and pulse.
PHSE	<phase>	:= {0 to 360}. The unit is °. When WVTP is noise, pulse wave, or DC, this parameter is invalid.

STDEV	<stdev>	:=Standard deviation of noise. The unit is "V". The valid range of parameter values can be found in the data sheet. Can only be set if WVTP is noise.
MEAN	<mean>	:=Mean value of noise. The unit is "V". The valid range of parameter values can be found in the data sheet. This parameter can only be set if WVTP is noise.
WIDTH	<width>	:=Positive pulse width. The unit is "s". The valid range of parameter values can be found in the data sheet. This parameter can only be set when WVTP is a pulse wave.
RISE	<rise>	:=Rise time (10%~90%). The unit is "s". The valid range of parameter values can be found in the data sheet. This parameter can only be set when WVTP is a pulse wave.
FALL	<fall>	:=Falling time (10%~90%). The unit is "s". The valid range of parameter values can be found in the data sheet. This parameter can only be set when WVTP is a pulse wave.
DLY	<delay>	:=Waveform delay. The valid range of parameter values can be found in the data sheet.
HLEV	<high level>	:=High level. The unit is "V". When WVTP is noise, this value has no effect.
LLEV	<low level>	:=Low level. The unit is "V". When WVTP is noise, this value has no effect.
BANDSTATE	<bandwidth switch>	:= {ON, OFF}. This parameter can only be set when WVTP is noise.
BANDWIDTH	<bandwidth value>	:=Noise bandwidth. The unit is "Hz". The valid range of parameter values can be found in the data sheet. This parameter can only be set if WVTP is noise.
LENGTH	<prbs length>	:= {3~32}. PRBS actual length=2^length-1. This parameter can only be set when WVTP is PRBS.
EDGE	<prbs rise/fall>	:= rise/fall time of PRBS. The unit is "s". The valid range of parameter values can be found in the data sheet. This parameter can only be set when WVTP is PRBS.
DIFFSTATE	<prbs differential switch>	:= {ON, OFF}. PRBS differential output mode. This parameter can only be set when WVTP is PRBS.

BITRATE	<prbs bit rate>	:= PRBS bitrate. The unit is "bps". The valid range of parameter values can be found in the data sheet. This parameter can only be set when WVTP is PRBS.
LOGICLEVEL	<prbslogiclevel rate>	:={TTL_CMOS,LVTTL_LVCMOS, ECL, LVPECL, LVDS, CUSTOM}. Used to set the logic level of PRBS. This parameter can only be set when WVTP is PRBS.
AMPVRMS	<amplitude>	:=Amplitude. Set the amplitude unit to Vrm.
AMPDBM	<amplitude>	:=Amplitude. Set the amplitude unit to dBm.
MAX_OUTPUT _AMP	<amplitude>	:=The maximum amplitude limit of the channel. You can check the valid range of parameter values in the data sheet.

QUERY SYNTAX	<channel>:BaSic_WaVe?
RESPONSE FORMAT	<channel>:BaSic_WaVe < parameter > < parameter >:={ All parameters of the current waveform}
EXAMPLE	Change the C1 waveform to a triangle wave: *C1:BSWV WVTP,RAMP* Change C1 frequency to 2000Hz: *C1:BSWV FRQ,2000* Set the amplitude of C1 to 3Vpp: *C1:BSWV AMP,3* Read the parameters of C1 from the device: *C1:BSWV?* Return: *C1:BSWV WVTP,SINE,FRQ,1000HZ,PERI,0.001S,AMP,4V, AMPVRMS,1.41421Vrms,MAX_OUTPUT_AMP,20V,OFST,0V, DLY,0S,DIFF_OFST,0V,HLEV,2V,LLEV,-2V,PHSE,0* Set the noise bandwidth of C1 to 100MHz: *C1:BSWV BANDWIDTH,100E6* or *C1:BSWV BANDWIDTH,100000000*

3.6.2 PRBS polynomial command

DESCRIPTION	This command is used to set or obtain PRBS custom polynomial parameters. This command is valid only when the basic waveform is PRBS.
COMMAND SYNTAX	<channel>:PRBS:< parameter > < value > < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
FSTAte	<state>	:={ON,OFF} Turn on or off PRBS custom polynomial switch.
FORMula	<value>	:=Set PRBS polynomial, double quotes are required.
SEED	<value>	:= {1, 2,...,M}, set the number of PRBS seeds.

QUERY SYNTAX	<channel>:PRBS:< parameter >?
EXAMPLE	Enable CH1 PRBS custom polynomial function: *C1:PRBS:FSTAte ON* Set the PRBS polynomial of CH1 to X7+X5+1: *C1:PRBS:FORMula "X7+X5"* Get the PRBS custom polynomial of CH1: *C1:PRBS:FORMula?* Return: *X7+X5+1*

3.6.3 DC final amplifier command

DESCRIPTION	This command sets or gets the final amplifier status.
COMMAND SYNTAX	<channel>:DcUFAmplifier < state > <channel>:={ C1, C2} < state >:={ ON, OFF}
QUERY SYNTAX	<channel>:DcUFAmplifier? <channel>:={ C1, C2}
RESPONSE FORMAT	<channel>: DcUFAmplifier,< state >

EXAMPLE	Turn on the final amplifier of CH1: *C1:DcUFAmplifier ON* Query the final amplifier status of CH2: *C2:DcUFAmplifier?* Return: *C2:DcUFAmplifier,OFF*

3.6.4 Harmonic command

DESCRIPTION	This command is used to set or obtain harmonic parameters. This command is only valid when the basic waveform is sine wave.
COMMAND SYNTAX	<channel>:HARMonic < parameter >,< value > < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
HARMSTATE	<STATE>	:={ON,OFF} turns harmonics on or off.
HARMTYPE	<TYPE>	:={EVEN,ODD,ALL} harmonic type.
HARMORDER	<NUM>	:={1, 2, ..., M}, where M is the maximum number of series supported.
HARMPHASE	<PHASE>	:={0~360}. The unit is °.
HARMAMP	<VALUE>	:=Specifies the amplitude of the harmonic. The range of values depends on the model. The unit is "Vpp".
HARMDBC	<VALUE>	:=Specifies the amplitude of the harmonic. The range of values depends on the model. The unit is "dBc".

QUERY SYNTAX	<channel>:HARMonic?
EXAMPLE	Enable the harmonic function of CH1: *C1:HARM HARMSTATE,ON* Set the harmonic series of CH1 to 2 and the amplitude to -6dBc: *C1:HARM HARMORDER,2,HARMDBC,-6* Get the harmonic parameters of CH1: *C1:HARM?*

	Return: *C1:HARM HARMSTATE,ON,HARMTYPE,EVEN,HARMORDER,2,HARMAMP, 0.0004V,HARMDBC,-80dBc,HARMPHASE,0*

3.6.5 Arbitrary waveform command

3.6.5.1 Switch arbitrary waveform command

DESCRIPTION	This command is used to set or read the type of arbitrary waveform.
COMMAND SYNTAX	Format 1:<channel>:ArbWaVe INDEX,<index> Format 2:<channel>:ArbWaVe NAME,<name> Format 3:<channel>:ArbWaVe NAME,<path> <channel>:={ C1, C2} <index>:= The index number of the arbitrary wave in the table below <name>:= Arbitrary wave names in the table below <path>:= Wave path, the path needs to add Local
QUERY SYNTAX	<channel>:ArbWaVe? <channel>:={ C1, C2}
RESPONSE FORMAT	<channel>:ARWV INDEX,<index>,NAME,<name>
EXAMPLE	Set the current waveform of CH1 through index number 2: *C1:ARWV INDEX,2* Read the current waveform of CH1: *C1:ARWV?* Return: *C1:ARWV INDEX,2,NAME,StairUp* Set the waveform of CH1 to heart wave through the waveform name: *C1:ARWV NAME,Cardiac* Set the waveform of CH1 through the waveform path: *C1:ARWV NAME,"Local/wave1.bin"*
Note	For the specific path, refer to the path in the file manager.
Related commands	3.14

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	ErfcInv	163	Duty38
11	ExpRise	62	Chebyshev1	113	ErfInv	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-Hann	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		

3.6.5.2 Low-resolution waveform command

DESCRIPTION	This command is used to set or obtain arbitrary waveform data.
COMMAND SYNTAX	<channel>:WVDT <parameter>,<value> <channel>:={ C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
WVNM	<name>	:= waveform name.
FRQ	<frequency>	:= frequency. The unit is "Hz".
AMP	<amplitude>	:= amplitude. The unit is "Vpp".
OFST	<offset>	:= offset. The unit is "V".
PHASE	<phase>	:= phase. The unit is °.
WAVEDATA	<data>	:= Waveform data. Data written to the waveform. The data is binary, and 16 bits are

		one waveform point. The codeword range of each waveform point is -32767~32767.
QUERY SYNTAX	Format 1:WVDT? Mn Format 2:WVDT? USER,<name> Format 3:WVDT? USER,<path>,<name> <path>:= Specify storage path <name>:= User-defined waveform name	
EXAMPLE	Write data 0x6000c0006000 into the 'wave1' waveform and send it to channel 1: *C1:WVDT WVNM,wave1,FRQ,2500,AMP,2.5,OFST,0.2, WAVEDATA, b'0x6000c0006000'*	

Note: It is recommended to use the method in 5.1.5 to write this instruction.

3.6.5.3 High-resolution waveform command

DESCRIPTION	This command can send the waveform of large data in segments to the specified bin file under the specified path.
COMMAND SYNTAX	<channel>:WVDT:SEGMENT WVNM,"wave1",<parameter>,<value> <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
WVNM	<name>	:= waveform name.
FRQ	<frequency>	:= frequency. The unit is "Hz". Only takes effect when writing starting data.
AMP	<amplitude>	:= amplitude. The unit is "Vpp". Only takes effect when writing starting data.
OFST	<offset>	:= offset. The unit is "V". Only takes effect when writing starting data.
PHASE	<phase>	:= phase. The unit is °. Only takes effect when writing starting data.
BEGIN,WAVEDATA	<data>	:= Waveform starting data. The data is binary, and 16 bits are one waveform point. The codeword range of each waveform point is -32767~32767.
WAVEDATA	<data>	:= Waveform intermediate data. The data is binary, and 16 bits are one waveform point. The codeword range of each waveform point is -

		32767~32767. At least a piece of intermediate data needs to be written.
END,WAVEDATA	<data>	:= Waveform end data. The data is binary, and 16 bits are one waveform point. The codeword range of each waveform point is -32767~32767.

EXAMPLE	Send a set of large data waveforms in segments to the local wave1.bin: *C1:WVDT:SEGMENT WVNM,"wave1",FRQ,2000,AMP,2,OFST,0.2,BEGIN,WAVEDATA,b'0x6000c0006'* *C1:WVDT:SEGMENT WVNM,"wave1",WAVEDATA,b'0x0006000'* *C1:WVDT:SEGMENT WVNM,"wave1",END,WAVEDATA,b'0x6000'*

Note: This command is suitable for large waveform data.When the waveform data is small,it is recommended to use the command in Chapter 3.15.9 to generate. It is recommended to use the method in 5.1.6 to write this instruction.

3.6.6 MOD command

DESCRIPTION	This command can set or get modulation waveform parameters.
COMMAND SYNTAX	<channel>:MoDulateWaVe <type> <channel>:MoDulateWaVe <parameter>,<value> <channel>:={C1, C2} <type>:={AM,DSBSC,FM,PM,PWM,ASK,FSK,PSK} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:= {ON, OFF}. Enable and disable modulation. If you want to set or read other parameters of modulation, you must first set STATE to "ON".
AM,SRC	<src>	:= {INT,EXT,CH1,CH2}. AM modulation source.
AM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. AM modulation waveform. This parameter can only be set when the modulation source is set to internal modulation.
AM,FRQ	<AM frequency>	:= AM frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when

		the modulation source is set to internal modulation.
AM,DEPTH	<depth>	:= {0 to 120}. AM Depth. The unit is "%". This parameter can only be set when the trigger source is set to internal trigger.
DSBSC,SRC	<src>	:= {INT, EXT, CH1, CH2}. DSBSC modulation source.
DSBSC,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. DSBSC modulation waveform. This parameter can only be set when the modulation source is set to internal modulation.
DSBSC,FRQ	<DSB-AM frequency>	:= DSBSC frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.
FM,SRC	<src>	:= {INT,EXT,CH1,CH2}. FM modulation source.
FM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. FM modulation waveform. This parameter can only be set when the modulation source is set to internal modulation.
FM,FRQ	<FM frequency>	:= FM frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.
FM,DEVI	<FM frequency deviation>	:= {0 to current carrier frequency}. FM frequency deviation. This value depends on the difference between the carrier frequency and the bandwidth frequency. This parameter can only be set when the modulation source is set to internal modulation.
PM,SRC	<src>	:= {INT,EXT,CH1,CH2}. PM modulation source.
PM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. PM modulation waveform. This parameter can only be set when the modulation source is set to internal modulation.
PM,FRQ	<PM frequency>	:= PM frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when

		the modulation source is set to internal modulation.
PM,DEVI	<PM phase offset>	:= {0 to 360}. PM phase deviation. The unit is °, and this parameter can only be set when the modulation source is set to internal modulation.
PWM,SRC	<src>	:= {INT,EXT,CH1,CH2}. PWM modulation source.
PWM,FRQ	<PWM frequency>	:= PWM frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.
PWM,DEVI	<PWM dev>	:=Duty cycle deviation. The unit is "s". The value depends on the pulse width of the carrier.
PWM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. PWM modulation waveform. This parameter can only be set when the modulation source is set to internal modulation.
ASK,SRC	<src>	:= {INT,EXT}. ASK modulation source.
ASK,KFRQ	< key frequency>	:= ASK keying frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.
FSK,SRC	<src>	:= {INT,EXT}. FSK modulation source.
FSK,KFRQ	< key frequency>	:= FSK keying frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.
FSK,HFRQ	<FSK_hop_freq>	:= FSK frequency hopping frequency. Same frequency as the base waveform. The unit is "Hz", and the valid range of parameter values can be found in the data sheet.
PSK,SRC	<src>	:= {INT,EXT}. PSK modulation source.
PSK,KFRQ	< key frequency>	:= PSK keying frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet. This parameter can only be set when the modulation source is set to internal modulation.

PSK,PLRT	<polarity>	:= {POS,NEG}.
CARR,WVTP	<wave type>	:={SINE, SQUARE, RAMP, ARB, PULSE}. Carrier frequency type.
CARR,FRQ	<frequency>	:= carrier frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet.
CARR,PHSE	<phase>	:= {0 to 360}. carrier phase. The unit is °.
CARR,AMP	<amplitude>	:=Carrier amplitude. The unit is "Vpp". The valid range of parameter values can be found in the data sheet.
CARR,OFST	<offset>	:=Carrier offset. The unit is volts. The valid range of parameter values can be found in the data sheet.
CARR,SYM	<symmetry>	:= {0 to 100}. When the carrier is RAMP, the carrier symmetry. The unit is "%".
CARR,DUTY	<duty>	:= {0 to 100}. When the carrier wave is a square wave or pulse wave, the carrier duty cycle. The unit is "%".
CARR,RISE	<rise>	:=When the carrier wave is a square wave or pulse wave, the rise time of the carrier wave. The unit is "s". The valid range of parameter values can be found in the data sheet.
CARR,FALL	<fall>	:=When the carrier wave is a square wave or pulse wave, the carrier wave fall time. The unit is "s". The valid range of parameter values can be found in the data sheet.
CARR,DLY	<delay>	:=When the carrier wave is a square wave or pulse wave, the carrier wave is delayed. The unit is "s". The valid range of parameter values can be found in the data sheet.

QUERY SYNTAX	<channel>:MoDulateWaVe?
RESPONSE FORMAT	<channel>:MDWV <parameter> <parameter>:={All parameters of the current modulation}
EXAMPLE	Set CH1 modulation status to open: *C1:MDWV STATE,ON* Set CH1 modulation type to AM: *C1:MDWV AM*

	Set AM modulation, and set the modulation waveform to sine wave: *C1:MDWV AM,MDSP,SINE* Read the modulation parameters of C1 whose status value is ON: *C1:MDWV?* Return: *C1:MDWVAM,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 the modulation parameters of CH1 whose status is OFF: *C1:MDWV?* Return: *C1:MDWV STATE,OFF* Set the FM frequency of CH1 to 1000Hz: *C1:MDWV FM,FRQ,1000* Set the carrier wave of CH1 to sine wave: *C1:MDWV CARR,WVTP,SINE* Set CH1 carrier frequency to 1000Hz: *C1:MDWV CARR,FRQ,1000*

3.6.7 SWEEP command

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

DESCRIPTION	This command is used to set or get the parameters of the scan waveform.
COMMAND SYNTAX	<channel>:SweepWaVe <para>,<value> <channel>:={C1, C2} <para>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:= {ON, OFF}. Enable and disable scanning. If you want to set or read other parameters of the scan, you must first set STATE to "ON".

TIME	<time>	:= Sweep time. The unit is "s". The valid range of parameter values can be found in the data sheet.
START	<start_freq>	:= Start frequency. Same as the basic waveform frequency. The unit is "Hz". The valid range of parameter values can be found in the data sheet.
STOP	<stop_freq>	:= End frequency. Same as the basic waveform frequency. The unit is "Hz". The valid range of parameter values can be found in the data sheet.
CENTER	<center_freq>	:= center frequency. The unit is "Hz". The valid range of parameter values can be found in the data sheet.
SPAN	<span_freq>	:= frequency span. The unit is "Hz". The valid range of parameter values can be found in the data sheet.
SWMD	<sweep_mode>	:= {LINE, LOG, STEP}, where LINE represents linear sweep, LOG represents logarithmic sweep, and STEP represents step sweep.
DIR	<direction>	:= {UP, DOWN, UP_DOWN}. Sweep direction.
SYM	<symmetry>	:= {0% to 100%}. Symmetry when the sweep direction is UP_DOWN.
TRSR	<trig_src>	:= {EXT, INT, MAN}. Trigger source. EXT represents external trigger, INT represents internal trigger, and MAN represents manual trigger.
MTRIG		:= Sends a manual trigger signal. This parameter is valid only when TRSR is MAN.
TRMD	<trig_mode>	:= {UP, DOWN, OFF}. Trigger output status. If TRSR is EXT, this parameter is invalid.
EDGE	<edge>	:= {RISE,FALL}. Available trigger edges. Valid only when TRSR is set to EXT.
STEPNUM	<num>	:= number of steps. The valid range of parameter values can be found in the data sheet.
CARR, WVTP	<wave type>	:= {SINE, SQUARE, RAMP, ARB}. Carrier type. If the carrier wave is pulse wave, noise, or DC. The waveform cannot be scanned.
CARR, FRQ	<frequency>	:= carrier frequency. The unit is "Hz", and the valid range of parameter values can be found in the data sheet.

CARR, PHSE	<phase>	:= {0 to 360}. carrier phase. The unit is °.
CARR, AMP	<amplitude>	:= Carrier amplitude. The unit is "Vpp". The valid range of parameter values can be found in the data sheet.
CARR, OFST	<offset>	:= Carrier offset. The unit is volts. The valid range of parameter values can be found in the data sheet.
CARR, SYM	<symmetry>	:= {0 to 100}. When the carrier is RAMP, the carrier symmetry. The unit is "%".
CARR, DUTY	<duty>	:= {0 to 100}. When the carrier wave is a square wave, the carrier duty cycle. The unit is "%".

QUERY SYNTAX	<channel>:SweepWaVe?
RESPONSE FORMAT	<channel>:SWWV <parameter> <parameter>:={All parameters of the current modulation}
EXAMPLE	Set the scanning status of CH1 to open: *C1:SWWV STATE,ON* Set the scan time of CH1 to 1 second: *C1:SWWV TIME,1* Set the stop frequency of CH1 to 1000Hz: *C1:SWWV STOP,1000* Set the trigger source of CH1 to manual: *C1:SWWV TRSR,MAN* Send a manual trigger signal to CH1: *C1:SWWV MTRIG* Read the scan parameters of CH1 whose status is ON: *C1:SWWV?* Return: *C1SWWV\sSTATE,ON,TIME,1S,STARTTIME,0S,ENDTIME,0S,ENDTIME,0S,STEPNUM,3,STOP,1500HZ,START,500HZ,CENTER,1000HZ,SPAN,1000HZ,TRSR,INT,TRMD,OFF,SWMD,LINE,DIR,UP,SYM,0,MARK_A_STATE,OFF,MARK_A_STEP,1HZ,MARK_B_STATE,OFF,MARK_B_STEP,1HZ,MARK_A_STATE,OFF,MARK_A_FREQ,1000HZ,MARK_B_STATE,OFF,MARK_B_FREQ,1000HZ,CARR,WVTP,SINE,FRQ,1000HZ,AMP,4V,AMPVRMS,1.41421Vrms,OFST,0V,PHSE,0*

	Read the scan parameters of CH1 whose status is OFF: *C1:SWWV?* Return: *C1:SWWV STATE,OFF*

3.6.7.2 <channel>:SWEep:TYPE <type>

DESCRIPTION	This command is used to set (query) the type of scan waveform.
COMMAND SYNTAX	<channel>:SWEep:TYPE <type> <channel>:={C1, C2} <type>:={FREQ,AMP,BOTH}
QUERY SYNTAX	<channel>:SWEep:TYPE? <channel>:={C1, C2}
EXAMPLE	Set the scan type of channel 1 to frequency scan: *C1:SWEep:TYPE FREQ* Query the scan type of channel 1: *C1:SWEep:TYPE?* Return: *"FREQ"*

3.6.7.3 <channel>:SWEep:AMODe <type>

DESCRIPTION	This command is used to set (query) the scan mode of amplitude scan.
COMMAND SYNTAX	<channel>:SWEep:AMODe <type> <channel>:={C1, C2} <type>:={LINE,STEP}
QUERY SYNTAX	<channel>:SWEep:AMODe? <channel>:={C1, C2}
EXAMPLE	Set the scan mode of channel 1 to step scan: *C1:SWEep:AMODe STEP* Query the scan mode of channel 1: *C1:SWEep:AMODe?* Return: *"STEP"*

3.6.7.4 <channel>:SWEep:ASNumber <value>

DESCRIPTION	This command is used to set (query) the number of steps for step amplitude scanning.
COMMAND SYNTAX	<channel>:SWEep:ASNumber <value> <channel>:={C1, C2} <value>:={An integer between 2 and 1024 }
QUERY SYNTAX	<channel>:SWEep:ASNumber? <channel>:={C1, C2}
EXAMPLE	Set the step scan number of channel 1 equal to 5: *C1:SWEep:ASNumber 5* Query the number of steps in the step scan mode of channel 1: *C1:SWEep:ASNumber?* Return: *"5"*

3.6.7.5 <channel>:SWEep:SHTime <value>

DESCRIPTION	This command is used to set (query) the starting hold time of scanning.
COMMAND SYNTAX	<channel>:SWEep:SHTime <value> <channel>:={C1, C2} <value>:= Floating point type value, unit seconds
QUERY SYNTAX	<channel>:SWEep:SHTime? <channel>:={C1, C2}
EXAMPLE	Set the starting hold time of channel 1 scan to 3 seconds: *C1:SWEep:SHTime 3* Query the starting hold time of channel 1 scan: *C1:SWEep:SHTime?* Return: *"3"*

3.6.7.6 <channel>:SWEep:EHTime <value>

DESCRIPTION	This command is used to set (query) the end holding time of scanning.
COMMAND SYNTAX	<channel>:SWEep:EHTime <value> <channel>:={C1, C2}

	<value>:= Floating point type value, unit seconds
QUERY SYNTAX	<channel>:SWEep:EHTime? <channel>:={C1, C2}
EXAMPLE	Set the end hold time of channel 1 scan to 3 seconds: *C1:SWEep:EHTime 3* Query the end holding time of channel 1 scan: *C1:SWEep:EHTime?* Return: *"3"*

3.6.7.7 <channel>:SWEep:RTime <value>

DESCRIPTION	This command is used to set (query) the return time after the scan ends.
COMMAND SYNTAX	<channel>:SWEep:RTime <value> <channel>:={C1, C2} <value>:= Floating point type value, unit seconds
QUERY SYNTAX	<channel>:SWEep:RTime? <channel>:={C1, C2}
EXAMPLE	Set the return time after channel 1 scan is completed to 3 seconds: *C1:SWEep:RTime 3* Query the return time after scanning of channel 1: *C1:SWEep:RTime?* Return: *"3"*

3.6.7.8 <channel>:SWEep:SAMPlitude <value>

DESCRIPTION	This command is used to set (query) the starting amplitude of the amplitude scan.
COMMAND SYNTAX	<channel>:SWEep:SAMPlitude <value> <channel>:={C1, C2} <value>:= Floating point type value in volts
QUERY SYNTAX	<channel>:SWEep:SAMPlitude? <channel>:={C1, C2}
EXAMPLE	Set the starting amplitude of channel 1 amplitude scan to 100mV:

	C1:SWEep:SAMPlitude 0.1 Query the starting amplitude of channel 1 amplitude scan: *C1:SWEep:SAMPlitude?* Return: *"0.1"*

3.6.7.9 <channel>:SWEep:EAMPlitude <value>

DESCRIPTION	This command is used to set (query) the termination amplitude of the amplitude scan.
COMMAND SYNTAX	<channel>:SWEep:EAMPlitude <value> <channel>:={C1, C2} <value>:= Floating point type value in volts
QUERY SYNTAX	<channel>:SWEep:EAMPlitude? <channel>:={C1, C2}
EXAMPLE	Set the stop amplitude of channel 1 amplitude scan to 900mV: *C1:SWEep:EAMPlitude 0.9* Query the termination amplitude of channel 1 amplitude scan: *C1:SWEep:EAMPlitude?* Return: *"0.9"*

3.6.7.10 <channel>:SWEep:CAMPlitude <value>

DESCRIPTION	This command is used to set (query) the center amplitude of the amplitude scan.
COMMAND SYNTAX	<channel>:SWEep:CAMPlitude <value> <channel>:={C1, C2} <value>:= Floating point type value in volts
QUERY SYNTAX	<channel>:SWEep:CAMPlitude? <channel>:={C1, C2}
EXAMPLE	Set the center amplitude of channel 1 amplitude scan to 500mV: *C1:SWEep:CAMPlitude 0.5* Query the center amplitude of channel 1 amplitude scan: *C1:SWEep:CAMPlitude?* Return: *"0.5"*

3.6.7.11 <channel>:SWEep:ASPan <value>

DESCRIPTION	This command is used to set (query) the amplitude range of the amplitude scan.
COMMAND SYNTAX	<channel>:SWEep:ASPan <value> <channel>:={C1, C2} <value>:= Floating point type value in volts
QUERY SYNTAX	<channel>:SWEep:ASPan? <channel>:={C1, C2}
EXAMPLE	Set the amplitude range of channel 1 amplitude scan to 800mV: *C1:SWEep:ASPan 0.8* Query the amplitude range of channel 1 amplitude scan: *C1:SWEep:ASPan?* Return: *"0.8"*

3.6.7.12 <channel>:SWEep:ADIRection <dir>

DESCRIPTION	This command is used to set (query) the scanning direction of amplitude scanning.
COMMAND SYNTAX	<channel>:SWEep:ADIRection <dir> <channel>:={C1, C2} <dir>:={UP,DOWN,UP_DOWN}
QUERY SYNTAX	<channel>:SWEep:ADIRection? <channel>:={C1, C2}
EXAMPLE	Set the scan direction of channel 1 amplitude scan downward: *C1:SWEep:ADIRection DOWN* Query the scanning direction of channel 1 amplitude scan: *C1:SWEep:ADIRection?* Return: *"DOWN"*

3.6.7.13 <channel>:SWEep:ASYMmetry <value>

DESCRIPTION	This command is used to set (query) the upper and lower scan symmetry of the amplitude linear scan.
COMMAND SYNTAX	<channel>:SWEep:ASYMmetry <value> <channel>:={C1, C2}

	<value>:= A value of type floating point.
QUERY SYNTAX	<channel>:SWEep:ASYMmetry? <channel>:={C1, C2}
EXAMPLE	Set the upper and lower scan symmetry of channel 1 amplitude scan to 80%: *C1:SWEep:ASYMmetry 80* Query the upper and lower scan symmetry of channel 1 amplitude scan: *C1:SWEep:ASYMmetry?* Return: *"80"*

3.6.8 BURST command

DESCRIPTION	This command is used to set or read pulse train waveform parameters.
COMMAND SYNTAX	<channel>:BursTWaVe <parameter>,<value> <channel>:={C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:= {ON, OFF}. Enable and disable bursts. If you want to set or read other parameters of the pulse train, you must first set STATE to "ON".
PRD	<period>	:=Burst period. The unit is "s". The valid range of parameter values can be found in the data sheet.
STPS	<start_phase>	:= {0 to 360}. The starting phase of the carrier wave. The unit is °. This value is invalid when the carrier wave is noise or pulse wave.
GATE_NCYC	<burst_mode>	:= {GATE,NCYC}. Burst mode. This value is invalid when the carrier is noise.
TRSR	<trig_src>	:= {EXT, INT, MAN} trigger source. EXT stands for external trigger.INT stands for internal trigger and MAN stands for manual trigger.
MTRIG		:=Send a manual trigger signal. This parameter is valid only when TRSR is MAN.
DLAY	<delay>	:=Trigger delay. The unit is "s". The valid range of parameter values can be found in the data sheet. This value is valid when GATE_NCYC is N cycle. This value is invalid when the carrier is noisy.
PLRT	<polarity>	:={NEG, POS}.Gate polarity, negative or positive.
TRMD	<trig_mode>	:= {RISE,FALL,OFF}. Trigger output mode. This parameter is valid when GATE_NCYC is an N cycle and the trigger source is internal trigger or manual trigger. This value is invalid when the carrier is noise.
EDGE	<edge>	:= {RISE,FALL}. Valid trigger edge. This value is valid when TRST is manual trigger or external trigger. This value is invalid when the carrier is noise.
TIME	<circle_ time>	:= {INF, 1, 2, ..., M}, the maximum number of N cycles supported by the M value depends on the model. INF sets the burst to infinite mode. Valid when GATE_NCYC is N cycle. This value is invalid when the carrier is noise.
COUNT	<counter>	:=Number of pulse trains, valid when the trigger source is external or manual. You can check the valid value range of the parameter in the data sheet.
CARR, WVTP	<wave type>	:= {SINE,SQUARE,RAMP,ARB,PULSE, NOISE}. Carrier waveform type.

CARR, FRQ	<frequency>	:=Carrier frequency. The unit is “Hz’”, and the valid range of parameter values can be found in the data sheet.
CARR, PHSE	<phase>	:= {0 to 360}. carrier phase. The unit is °.
CARR, AMP	<amplitude>	:=Carrier amplitude. The unit is "Vpp". The valid range of parameter values can be found in the data sheet.
CARR, OFST	<offset>	:=Carrier offset. The unit is "V". The valid range of parameter values can be found in the data sheet.
CARR, SYM	<symmetry>	:= {0 to 100}. Carrier symmetry, this value is valid when the carrier is a triangular wave. The unit is "%".
CARR, DUTY	<duty>	:= {0 to 100}. Carrier duty cycle, this value is valid when the carrier wave is square wave and pulse wave. The unit is "%".
CARR, RISE	<rise>	:=rise time. This value is valid when the carrier wave is pulse wave. The unit is "s". The valid range of parameter values can be found in the data sheet.
CARR, FALL	<fall>	:=Falling time. This value is valid when the carrier wave is pulse wave. The unit is "s". The valid range of parameter values can be found in the data sheet.
CARR, DLY	<delay>	:=Pulse wave delay. This value is valid when the carrier wave is pulse wave. The unit is "s". The valid range of parameter values can be found in the data sheet.
CARR, STDEV	<stdev>	:=Standard deviation of noise. The unit is "V". The valid range of parameter values can be found in the datasheet.
CARR, MEAN	<mean>	:=Mean value of noise. The unit is "V". The valid range of parameter values can be found in the data sheet.

QUERY SYNTAX	<channel>:BursTWaVe?
RESPONSE FORMAT	<channel>:BTWV <parameter> <parameter>:={ All parameters of the current pulse train}
EXAMPLE	Set CH1 burst status to ON: *C1:BTWV STATE,ON* Set CH1 burst period to 1s: *C1:BTWV PRD,1* Set burst delay to 1s: *C1:BTWV DLAY,1*

	Set CH1 burst cycle to infinite: *C1:BTWV TIME,INF* Read the CH2 pulse train parameters whose status 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 the CH2 pulse train parameters whose status is OFF: *C2:BTWV?* Return: *C2:BTWV STATE,OFF*

DESCRIPTION	This command is used to set or read the pulse train idle level parameters.
COMMAND SYNTAX	<channel>:BURSt:HVALue <parameter> <channel>:={C1, C2} <parameter>:={ZERo, AVELue, EVALue}
QUERY SYNTAX	<channel>:BURSt:HVALue?
EXAMPLE	Set the idle level of CH1 pulse train to the starting value: *C1:BURSt:HVALue SVALue* Read the idle level of CH1 pulse train: *C1:BURSt:HVALue?* Return: *SVALue*

3.7 AWG command

DESCRIPTION	This command is used to set or get AWG parameters.
COMMAND SYNTAX	<channel>:AWG:< parameter >,<value> <channel>:={C1, C2} < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:={RUN, STOP}, turn on or off the AWG running state.
SRATE	<value>	:=Set the AWG sampling rate, the unit is "Sa/s".
INTPtype	<type>	:={ZERO,LINEAR,SINC,SINC13,SINC27}, set the interpolation type of AWG.
HOLDtype	<type>	:= {MID, START, END, USER}, set the idle level type of AWG.
USRHOLD	<value>	:=Set the user-defined idle level value, the unit is V.
DEFAult	None	:=Set the default settings for AWG.
SCALe	<value>	:=Set the AWG amplitude ratio, the unit is %.
OFFset	<value>	:=Set AWG offset, unit is V.
INCReasing	<type>	:= {INT, ZERO, HLAS, DUPL}, set the interpolation method of AWG.
DECReasing	<type>	:= {DECi, CTAi, CHEa}, set the extraction method of AWG.
RMODe	<type>	:= {CONT, TCON, BURS, STEP, ADV}, set the AWG run mode.
TRIGger:SOURce	<type>	:= {MAN,TIMe,EXT},Set the trigger source type of AWG.
TRIGger:IMMediate	None	:=Set the manual trigger button of AWG.
TRIGger:TIMer	<value>	:=Set the AWG timer time, the unit is S.
TRIGAger:SLOPe	<type>	:= {RISe,FALL,BOTH},Set the external trigger edge of AWG.
TRIGger:DELAy	<value>	:=Set the trigger delay of AWG, the unit is S.
BURSt:COUNt	<value>	:=Set the number of pulse trains for AWG.

BURSt:PERIod	<value>	:=Set the Burst period of AWG, the unit is S.
SEGMent:STARTNumb	<value>	:=Set the starting segment of AWG Segment.
SEGMent:INSErt	<index>	:=Insert a Segment at the index position.
SEGMent:DELEte	<index>	:=Delete the Segment at index position.
SEGMent:CLEAR	None	:=Clear the sequence and keep only the default Segment.
SAVE	<path>	:=Save the AWG waveform of the channel, <path>:=PATH,"Local/xxx.AWGx",SWFS,<swfs> <swfs>:={"TRUE","FALSE"}
LOAD	<path>	:=Load the AWG waveform of the channel, <path>:="Local/xxx.AWGx"
SEGMent#:AMPlitude	<value>	:=Set the Segment amplitude of AWG, the unit is V, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:OFFset	<value>	:=Set the Segment offset of AWG, the unit is V, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:VOLTage:HIGH	<value>	:=Set the high level of Segment of AWG, the unit is V, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:VOLTage:LOW	<value>	:=Set the low level of Segment of AWG, the unit is V, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:LOOP	<value>	:=Set the Segment loop of AWG, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:GOTO	<value>	:=Set the Segment goto of AWG, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:LENGth	<length>	:=Set the Segment waveform length of AWG, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:WAVeform	<wave>	:={Sine, Noise,...} Set the Segment waveform data source of AWG. For specific data sources, see Chapter 3.6.5.1, #:={1, 2,...,M}, indicates the Segment number.
SEGMent#:WAITEvent	<value >	:= {AUTO, MAN, EXT, TIMe}, set the waiting event of the advanced operating mode of AWG, #:={1, 2,...,M}, indicates the Segment number.

QUERY SYNTAX	<channel>:AWG:<parameter>?
EXAMPLE	Set up AWG run for channel 1:

C1:AWG:STATE RUN

Set AWG default settings for channel 1:
C1:AWG:DEFAult

Set the AWG amplitude ratio of channel 1 to 50%:
C1:AWG:SCAle 0.5

Set the AWG operating mode of channel 1 to advanced:
C1:AWG:RMODe ADV

Set channel 1 to save the waveform file "Local/aaa,AWGx":
C1:AWG:SAVE PATH,"Local/aaa.AWGx",SWFS,"TRUE"

Set channel 1 to load the waveform file "Local/aaa,AWGx":
C1:AWG:LOAD "Local/aaa.AWGx"

Set the data source of Segment-2 of channel 1 to Sine:
C1:AWG: SEGMent2:WAVeform Sine

3.8 IQ command

3.8.1 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*

3.8.2 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*

3.8.3 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*

3.8.4 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

3.8.5 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*

3.8.6 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

3.8.7 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*

3.8.8 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*

3.8.9 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*

3.8.10 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

3.8.11 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
16QAM_2				

3.8.12 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 user's local storage waveform wave1.arb: *IQ:WAVEload:USERstored "Local/wave1.arb"* Set the I/Q waveform to the network storage waveform wave1.arb: *IQ:WAVEload:USERstored "net_storage/wave/wave1.arb"* Set the I/Q waveform to the U disk storage waveform wave1.arb: *IQ:WAVEload:USERstored "U-disk0/ wave/wave1.arb"*

3.8.13 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*

3.9 Frequency hopping command

3.9.1 <channel>:FHOP:SWITch

DESCRIPTION	This command is used to set the frequency hopping function switch status.
COMMAND SYNTAX	<channel>:FHOP:SWITch <state> <channel>:={C1，C2} <state>:= {ON，OFF}
QUERY SYNTAX	<channel>:FHOP:SWITch?
EXAMPLE	Channel 1 turns on frequency hopping: *C1:FHOP:SWITch ON* Query channel 1 frequency hopping switch status: *C1:FHOP:SWITch?* Return: *"ON"*

3.9.2 <channel>:FHOP:TYPE

DESCRIPTION	This command is used to set the frequency hopping mode.
COMMAND SYNTAX	<channel>:FHOP:TYPE <type> <channel>:={C1，C2} <type>:= {MANUAL,RHOP,RLIST}
QUERY SYNTAX	<channel>:FHOP:TYPE?
EXAMPLE	Set channel 1 frequency hopping to list mode: *C1:FHOP:TYPE MANUAL* Query channel 1 frequency hopping mode: *C1:FHOP:TYPE?* Return: *"MANUAL"*

3.9.3 <channel>:FHOP:TIME

DESCRIPTION	This command is used to set the duration of frequency hopping on each frequency point.

COMMAND SYNTAX	<channel>:FHOP:TIME <time> <channel>:={C1，C2} <time>:= {Floating point number from 0 to 1000}
QUERY SYNTAX	<channel>:FHOP:TIME?
EXAMPLE	Set the duration of channel 1 frequency hopping at each frequency point to 1ms: *C1:FHOP:TIME 0.001* Query the duration of channel 1 frequency hopping at each frequency point: *C1:FHOP:TIME?* Return: *"0.001"*

3.9.4 <channel>:FHOP:SFREquency

DESCRIPTION	This command is used to set the starting frequency of random frequency hopping.
COMMAND SYNTAX	<channel>:FHOP:SFREquency <freq> <channel>:={C1，C2} <freq>:= {A floating point number that does not exceed the device bandwidth}
QUERY SYNTAX	<channel>:FHOP:SFREquency?
EXAMPLE	Set the starting frequency of channel 1 random frequency hopping to 1MHZ: *C1:FHOP:SFREquency 1000000* Query the starting frequency of random frequency hopping of channel 1: *C1:FHOP:SFREquency?* Return: *"1000000"*

3.9.5 <channel>:FHOP:EFREquency

DESCRIPTION	This command is used to set the end frequency of random frequency hopping.

COMMAND SYNTAX	<channel>:FHOP:EFREquency <freq> <channel>:={C1，C2} <freq>:= {A floating point number that does not exceed the device bandwidth}
QUERY SYNTAX	<channel>:FHOP:EFREquency?
EXAMPLE	Set the end frequency of channel 1 random frequency hopping to 100MHZ: *C1:FHOP:EFREquency 100000000* Query the end frequency of random frequency hopping of channel 1: *C1:FHOP:EFREquency?* Return: *"100000000"*

3.9.6 <channel>:FHOP:FSTep

DESCRIPTION	This command is used to set the frequency step of random frequency hopping.
COMMAND SYNTAX	<channel>:FHOP:FSTep <freq> <channel>:={C1，C2} <freq>:= {A floating point number that does not exceed the device bandwidth}
QUERY SYNTAX	<channel>:FHOP:FSTep?
EXAMPLE	Set the frequency step of channel 1 random frequency hopping to 10MHZ: *C1:FHOP:FSTep 10000000* Query the frequency step of random frequency hopping of channel 1: *C1:FHOP: FSTep?* Return: *"10000000"*

3.9.7 <channel>:FHOP:RPATtern

DESCRIPTION	This command is used to set the pseudo-random pattern of random frequency hopping.

COMMAND SYNTAX	<channel>:FHOP:RPATtern <pattern> <channel>:={C1，C2} <pattern>:= {an integer from 3 to 32}
QUERY SYNTAX	<channel>:FHOP:RPATtern?
EXAMPLE	Set the pseudo-random pattern of channel 1 random frequency hopping to PRBS-7: *C1:FHOP:RPATtern 7* Query the pseudo-random pattern of random frequency hopping of channel 1: *C1:FHOP:RPATtern?* Return: *"7"*

3.9.8 <channel>:FHOP:RLPAttern

DESCRIPTION	This command is used to set the pseudo-random pattern of frequency hopping random list mode.
COMMAND SYNTAX	<channel>:FHOP:RLPAttern <pattern> <channel>:={C1，C2} <pattern>:= {an integer from 3 to 32}
QUERY SYNTAX	<channel>:FHOP:RLPAttern?
EXAMPLE	Set the pseudo-random pattern of channel 1 frequency hopping list mode to PRBS-7: *C1:FHOP:RLPAttern 7* Query the pseudo-random pattern of channel 1 frequency hopping list mode: *C1:FHOP:RLPAttern?* Return: *"7"*

3.9.9 <channel>:FHOP:ALSTate

DESCRIPTION	This command is used to set the enable status of the random frequency hopping frequency avoidance list.
COMMAND SYNTAX	<channel>:FHOP:ALSTate <state>

	<channel>:={C1 , C2} <state>:= {ON, OFF}
QUERY SYNTAX	<channel>:FHOP:ALSTate?
EXAMPLE	Frequency filtering table to enable random frequency hopping for channel 1: *C1:FHOP:ALSTate ON* Query the enable status of the random frequency hopping frequency filter table of channel 1: *C1:FHOP:ALSTate?* Return: *"ON"*

3.9.10 <channel>:FHOP:AFLIst

DESCRIPTION	This command is used to insert an item at the specified position in the frequency table.
COMMAND SYNTAX	<channel>:FHOP:AFLIst <index>,<freq> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096} <freq>:= {A floating point number that does not exceed the device bandwidth}
EXAMPLE	Channel 1 inserts an item in the third item of the frequency table, the frequency is 1KHz: *C1:FHOP:AFLIst 3,1000*

3.9.11 <channel>:FHOP:DFLIst

DESCRIPTION	This command is used to delete the item at the specified position in the frequency table.
COMMAND SYNTAX	<channel>:FHOP:DFLIst <index> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096}
EXAMPLE	Delete the third item of the channel 1 frequency table: *C1:FHOP:DFLIst 3*

3.9.12 <channel>:FHOP:CFLIst

DESCRIPTION	This command is used to clear the frequency table (restore to default values).
COMMAND SYNTAX	<channel>:FHOP:CFLIst <channel>:={C1 , C2}
EXAMPLE	Clear the frequency table of channel 1: *C1:FHOP:CFLIst*

3.9.13 <channel>:FHOP:MFLIst

DESCRIPTION	This command is used to modify the item at the specified position in the frequency table to the set value.
COMMAND SYNTAX	<channel>:FHOP:MFLIst <index>,<freq> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096} <freq>:= {A floating point number that does not exceed the device bandwidth}
EXAMPLE	Modify the frequency of the third item of the channel 1 frequency table to 2KHz: *C1:FHOP:MFLIst 3,2000*

3.9.14 <channel>:FHOP:AOLIst

DESCRIPTION	This command is used to insert an item at the specified position in the order list.
COMMAND SYNTAX	<channel>:FHOP:AOLIst <index>,<freq_num> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096} <freq_num>:= {An integer from 1 to 4096, not exceeding the length of the frequency table}
EXAMPLE	Channel 1 inserts an item in the third item of the order list, the frequency number is 4: *C1:FHOP:AOLIst 3,4*

3.9.15 <channel>:FHOP:DOLIst

DESCRIPTION	This command is used to delete the item at the specified position in the order list.
COMMAND SYNTAX	<channel>:FHOP:DOLIst <index> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096}
EXAMPLE	Delete the third item in the channel 1 order list: *C1:FHOP:DOLIst 3*

3.9.16 <channel>:FHOP:COLIst

DESCRIPTION	This command is used to clear the order list (restore to default value).
COMMAND SYNTAX	<channel>:FHOP:COLIst <channel>:={C1 , C2}
EXAMPLE	Clear the order list of channel 1: *C1:FHOP:COLIst*

3.9.17 <channel>:FHOP:MOLIst

DESCRIPTION	This command is used to modify the item at the specified position in the order list to the set value.
COMMAND SYNTAX	<channel>:FHOP:MOLIst <index>,<freq_num> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096} <freq_num>:= {An integer from 1 to 4096, not exceeding the length of the frequency table}
EXAMPLE	Modify the value of the third item in the order list of channel 1 to 8: *C1:FHOP:MOLIst 3,8*

3.9.18 <channel>:FHOP:AALIst

DESCRIPTION	This command is used to insert an item at the end of the frequency avoidance list.
COMMAND SYNTAX	<channel>:FHOP:AALIst <start_freq>,<end_freq> <channel>:={C1 , C2}

	<start_freq>:= { A floating point number that does not exceed the device bandwidth and is less than end_freq} <end_freq>:= { A floating point number that does not exceed the device bandwidth and is greater than start_freq}
EXAMPLE	Channel 1 inserts an item at the end of the frequency avoidance list with a start frequency of 1KHz and an end frequency of 5KHz: *C1:FHOP:AALIst 1000,5000*

3.9.19 <channel>:FHOP:DALIst

DESCRIPTION	This command is used to delete the specified item in the frequency avoidance list.
COMMAND SYNTAX	<channel>:FHOP:DALIst <channel>:={C1 , C2}
EXAMPLE	Remove item 3 from channel 1 frequency avoid list: *C1:FHOP:DALIst 3*

3.9.20 <channel>:FHOP:CALIst

DESCRIPTION	This command is used to clear the frequency avoidance list.
COMMAND SYNTAX	<channel>:FHOP:CALIst <channel>:={C1 , C2}
EXAMPLE	Clear frequency avoidance list for channel 1: *C1:FHOP:CALIst*

3.9.21 <channel>:FHOP:MALIst

DESCRIPTION	This command is used to modify the setting value of the specified position in the frequency avoidance list.
COMMAND SYNTAX	<channel>:FHOP:MALIst <index>,<start_freq>,<end_freq> <channel>:={C1 , C2} <index>:= {Integer from 1 to 4096} <start_freq>:= { A floating point number that does not exceed the device bandwidth and is less than end_freq} <end_freq>:= { A floating point number that does not exceed the device bandwidth and is greater than start_freq}

EXAMPLE	Modify the start frequency of the third item in the frequency avoidance list of channel 1 to 1KHz and the end frequency to 5KHz: *C1:FHOP:MALIst 3,1000,5000*

3.9.22 <channel>:FHOP:LFLIst

DESCRIPTION	This command is used to load a frequency table from a file.
COMMAND SYNTAX	<channel>:FHOP:LFLIst <file> <channel>:={C1 , C2} <file>:= File name (including path)
EXAMPLE	Load the frequency table of channel 1 from the file freq.hop: *C1:FHOP:LFLIst "freq.hop"*

3.9.23 <channel>:FHOP:SFLIst

DESCRIPTION	This command is used to save the current frequency table to a file.
COMMAND SYNTAX	<channel>:FHOP:SFLIst <file> <channel>:={C1 , C2} <file>:= File name (including path)
EXAMPLE	The frequency table of channel 1 is saved to the file freq.hop: *C1:FHOP:SFLIst "freq.hop"*

3.9.24 <channel>:FHOP:LOLIst

DESCRIPTION	This command is used to load the order list from a file.
COMMAND SYNTAX	<channel>:FHOP:LOLIst <file> <channel>:={C1 , C2} <file>:= File name (including path)
EXAMPLE	Load the order list for channel 1 from the file order.hop: *C1:FHOP:LOLIst "order.hop"*

3.9.25 <channel>:FHOP:SOLIst

DESCRIPTION	This command is used to save the current order list to a file.

COMMAND SYNTAX	<channel>:FHOP:SOLIst <file> <channel>:={C1，C2} <file>:= File name (including path)
EXAMPLE	The order list of channel 1 is saved to the file order.hop: *C1:FHOP:SOLIst "order.hop"*

3.9.26 <channel>:FHOP:LALIst

DESCRIPTION	This command is used to load the frequency avoidance list from a file.
COMMAND SYNTAX	<channel>:FHOP:LALIst <file> <channel>:={C1，C2} <file>:= File name (including path)
EXAMPLE	Load frequency avoidance list for channel 1 from file avoid.hop: *C1:FHOP:LALIst "avoid.hop"*

3.9.27 <channel>:FHOP:SALIst

DESCRIPTION	This command is used to save the current frequency avoidance list to a file.
COMMAND SYNTAX	<channel>:FHOP:SALIst <file> <channel>:={C1，C2} <file>:= File name (including path)
EXAMPLE	The frequency avoidance list for channel 1 is saved to the file avoid.hop: *C1:FHOP:SALIst "avoid.hop"*

3.10 Multi Tone command

DESCRIPTION	This command is used to set or get multitone parameters.
COMMAND SYNTAX	Format1: <channel>:MTONE:<parameter>,<value> <channel>:={C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter} Format2: <channel>:MTONE:<parameter> <index>,<value1>,<value2> <channel>:={C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STAte	<state>	:={ON,OFF}, turn on or off the multi-tone function.
RSTAte	<state>	:={ON,OFF}, turn on or off the multi-tone running state.
NSTAte	<state>	:={ON,OFF}, turn on or off the Notch List state.
SRATe	<value>	:=Set the sampling rate, the unit is "Sa/s".
AMPlitude	<value>	:=Set the amplitude. The unit is "V".
SFREquency	<value>	:=Set the starting frequency. The unit is "HZ".
EFREquency	<value>	:=Set the end frequency. The unit is "HZ".
TNUMber	<value >	:=Set the Tone Number.
ANLIst	<value1>,<value2>	:=Set the Notch List to add an item.
DNLIst	<index>	:=Set the Notch List to delete the index item.
CNLIst	None	:=Clear the Notch List.
MNLIst	<index>,<value1>,<value2>	:=Set the Notch List to modify the index item.
NLISt	None	:=Query Notch List.
ATLIst	<value>	:=Set the Tone List to add an item.
DTLIst	<index>	:=Set the Tone List to delete the index item.

CTLIst	None	:=Clear the Tone List.
STLIst	None	:=Set the Tone List sorting.
MTLIst	<index>,<value>	:=Set the Tone List to modify the index item.
TLISt	None	:=Query the Tone List.
AMTList	None	:=Add the set frequency point to the Tone List.

QUERY SYNTAX	<channel>:MTONE:<parameter>?
EXAMPLE	Set up multi-tone run for channel 1: *C1:MTONE:RSTAte ON* Get the number of multi-tone of channel 1: *C1:MTONE:TNUMber?* Return: *2* Set channel 1 multi-tone Notch List to add an item, starting value 1000Hz, end value 2000Hz: *C1:MTONE:ANList 1000,2000* Set channel 1 multi-tone Notch List to delete item 2: *C1:MTONE:DNList 2* Modify 1 multi-tone Notch List item 1 starting value 3000Hz, end value 4000Hz: *C1:MTONE:MNList 1,3000,4000* Clear the channel 1 multi-tone Notch List: *C1:MTONE:CNList* Set channel 1 multi-tone Tone List to add an item with a frequency value of 2000Hz: *C1:MTONE:ATList 2000* Sort all frequency points in the multi-tone Tone List of channel 1: *C1:MTONE:STLISt* Get all frequency points in the multi-tone Tone List of channel 1: *C1:MTONE:TLISt?* Return: *1,1000.000000;2,10000.000000* Set channel 1 multi-audio point and add it to the list: *C1:MTONE:AMTList*

3.11 Multi Pulse command

DESCRIPTION	This command is used to set or obtain multi-pulse parameters.
COMMAND SYNTAX	Format1: <channel>:MPULse:<parameter>,<value> <channel>:={C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter} Format2: <channel>:MPULse:<parameter> <value1>,<value2>,<value3> <channel>:={C1, C2} <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STAte	<state>	:={ON,OFF}, turn on or off the multi-pulse function.
RSTAte	<state>	:={ON,OFF}, turn on or off the multi-pulse running state.
PNUMber	<count>	:={2, 3,...,30}, specify the number of pulses.
HLEVel	<value>	:=High level. The unit is "V".
LLEVel	<value>	:=Low level. The unit is "V".
TRIGger:SOURce	<type>	:={INT,EXT,MAN,TIMER}, specify the trigger source of multi-pulse.
TRIGger:TIMer	<value >	:=Timing time when timer triggers.
TRIGger:DELAy	<value >	:=Trigger delay when triggered externally or manually.
MPULse	<value1>,<value2>,<value3>	:=Set the pulse width and gap of the pulse number.
SRate	<value>	:=Set the value of the sampling rate, valid when the sampling rate type is custom.
SRate:TYPe	<type>	:={AUTO,CUSTOM}, set the sampling rate type.

QUERY SYNTAX	<channel>:MPULse:<parameter>?

EXAMPLE	Set up multi-pulse operation for channel 1: *C1:MPULse:RSTAte ON* Get the number of pulses of channel 1: *C1:MPULse:PNUMber?* Return: *2* Set manual trigger for channel 1 once: *C1:AWG:Trigger* Set the pulse width of channel 1 pulse 1 to 1ms and the gap to 2ms: *C1:MPULse:MPULse 1,0.001,0.002* Get the pulse width and gap of all pulses on channel 1: *C1:MPULse:PULse?* Return: *1,0.001000,0.002000;2,0.000001,0.000004* Set channel 1 multi-pulse sampling rate type to custom: *C1:MPULse:SRate:TYPe CUSTOM* Set channel 1 multi-pulse sampling rate to 200MSa/s: *C1:MPULse:SRate 200000000* Get channel 1 multi-pulse sampling rate: *C1:MPULse:SRate?* Return: *200000000*

3.12 Counter command

DESCRIPTION	This command sets or gets counter parameters.
COMMAND SYNTAX	SENSe:COUNTer:CONFig:<parameter>,<value> <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:= {ON, OFF}. Counter status.
MODE	<mode>	:= {FREQuency,TOTALizer},Frequency mode and Totalizer mode.
COUPLing	<mode>	:= {AC, DC}, coupled mode.
HFREJect	<HFR>	:= {OFF, ON} or {0, 1}, high frequency suppression state.
TLEVel	<triglev>	:=Trigger level. The range of valid values depends on the model.The unit is "V".
SEXIT	<mode>	:= { OFF, ON} or {0, 1},Turn the counter on or off when exiting.
PAUSe	<state>	:= { OFF, ON}, pause switch.
MEASure	< type >	:={FREQ,PERIOD,DUTY_CYCLe},frequency, period, duty cycle. Valid in frequency mode.
GATE:STATe	<type>	:={OFF, ON} or {0, 1} to turn on or off. Valid in totalizer mode.
EDGE	<edge>	:={RISE, FALL}, rising edge or falling edge. Valid in totalizer mode.
GATE:MODE	<mode>	:={LEVEL, AFTER_EDGE}, level or post-edge mode. Valid in totalizer mode.
GATE:POLarity	<polarity>	:={NEGative, POSitive}, negative or positive polarity. Valid in totalizer mode.
GATE:EDGE	<edge>	:={RISE, FALL}, rising edge or falling edge. Valid in totalizer mode.

QUERY SYNTAX	SENSe:COUNTer:CONFig:<parameter>?
EXAMPLE	Open counter: *SENSe:COUNTer:CONFig:STATE ON* Set frequency counter mode: *SENSe:COUNTer:CONFig:MODE FREQuency*

DESCRIPTION	This command sets the measurement parameters in frequency counter mode.
COMMAND SYNTAX	SENSe:COUNTer:FREQuency:<parameter> <value> <parameter>:={MEAN,MAX,MIN,SDEV,SNUMber,FDEViation, RFREQuency}}
QUERY SYNTAX	SENSe:COUNTer:FREQuency:<parameter>?
EXAMPLE	Set the measurement type in frequency mode to period: *SENSe:COUNTer:FREQuency:MEASure PERIOD* Query the frequency results measured in frequency mode: *SENSe:COUNTer:FREQuency?* Query the duty cycle result measured in frequency mode: *SENSe:COUNTer:FREQuency:DUTY?* Query the frequency deviation results measured in frequency mode: *SENSe:COUNTer:FREQuency:FDEViation?* Query the frequency average deviation result measured in frequency mode: *SENSe:COUNTer:FREQuency:MEAN:FDEViation?* Query the average value of the duty cycle measured in frequency mode: *SENSe:COUNTer:FREQuency:DUTY:MEAN?* Query the reference frequency setting in frequency counter mode: *SENSe:COUNTer:FREQuency:RFREQuency?*

DESCRIPTION	This command sets the measurement parameters in totalizer mode.
COMMAND SYNTAX	SENSe:COUNTer:TOTalizer:<parameter> <value> <parameter>:={ Parameters in the table below} <value>:={ The value of the relevant parameter}
QUERY SYNTAX	SENSe:COUNTer:TOTalizer?
EXAMPLE	Set the counter gate control mode in totalizer mode to level mode: *SENSe:COUNTer:TOTalizer:GATE:MODE LEVEL*

DESCRIPTION	This command clears the current measurement data of the counter.
COMMAND SYNTAX	SENSe:COUNTer:CLEar

3.13 System setup command

3.13.1 Sync signal command

DESCRIPTION	This command is used to set the synchronization signal.
COMMAND SYNTAX	<channel>:SYNC <state> <state>:={ON,OFF}
QUERY SYNTAX	<channel>:SYNC?
RESPONSE FORMAT	<channel>:SYNC <state>
EXAMPLE	Turn on sync: *C1:SYNC ON* Read synchronization function status: *C1:SYNC?* Return: *C1:SYNC ON,TYPE,CH1*

DESCRIPTION	This command is used to set the synchronization signal type.
COMMAND SYNTAX	<channel>:SYNC <parameter>,<value> <parameter>:=TYPE <value>:={CH,MOD_CH}
EXAMPLE	Output CH1 synchronization signal: *C1:SYNC TYPE,CH*

3.13.2 Clock source command

DESCRIPTION	This command sets or gets the clock source, or sets the 10MHz clock output.
COMMAND SYNTAX	Format1: ROSCillator <src> <src>:={INT,EXT} Format2: ROSCillator 10MOUT,<state> <state>:={ON,OFF}, Clock output status.
QUERY SYNTAX	ROSCillator?
RESPONSE FORMAT	ROSC <src>,10MOUT,<state>
EXAMPLE	Set the internal time base as the clock source: *ROSC INT* Turn on 10MHz clock output: *ROSC 10MOUT,ON*

3.13.3 Power-on setting command

DESCRIPTION	Set or get the power-on boot mode.
COMMAND SYNTAX	Format1: Sys_CFG <mode> <mode>:={DEFAULT, LAST} Format2: Sys_CFG <mode>,<path> <mode>:=USER <path>:= The path to the configuration file of user storage (local, network storage, USB flash drive), including file name and suffix.
QUERY SYNTAX	Sys_CFG?
RESPONSE FORMAT	SCFG <mode> SCFG <mode>,<path>
EXAMPLE	Set power-on to the last time: *SCFG LAST* Set power-on recovery file: *SCFG USER,"Local/state.xml"*

3.13.4 Multi-device synchronization command

DESCRIPTION	This command sets synchronization between two or more instruments and achieves in-phase output.
COMMAND SYNTAX	CASCADE < parameter >,< value > < parameter >:={ Parameters in the table below} < value >:={ The value of the relevant parameter}

parameter	value	describe
STATE	<state>	:={ON,OFF}, Turn on or off multi-device synchronization.
MODE	<type>	:= {MASTER, SLAVE}
DELAY	<time>	:= {0 to 0.000025}, unit = s, this parameter can only be set in slave mode.
SYNC_PHASE		

QUERY SYNTAX	<channel>:FILTer?
EXAMPLE	Set the digital filter of CH1 to turn on: *C1:FILT STATE,ON*

3.13.5 Language command

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

3.13.6 Buzzer command

DESCRIPTION	This command is used to turn 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*

3.13.7 Key command

DESCRIPTION	This command is used to turn on or off the front panel keys.
COMMAND SYNTAX	KEY <state> <state>:={ON, OFF}
QUERY SYNTAX	KEY?
RESPONSE FORMAT	KEY <state>
EXAMPLE	Open the front panel buttons: *KEY ON*

3.13.8 Screen saver command

DESCRIPTION	This command is used to turn off or set the screen saver time (unit: minutes).
COMMAND SYNTAX	SCreen_SaVe <parameter> <parameter>:={OFF, 1, 5, 15, 30, 60}
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*

3.13.9 IP command

DESCRIPTION	This parameter is used to set or read the IP address of the device.
COMMAND SYNTAX	SYSTem:COMMunicate:LAN:IPADdress <parameter> <parameter>:=<value1>.<value2>.<value3>.<value4> <value1>:={an integer value between 1 and 223} <value2>:={an integer value between 0 and 255} <value3>:={an integer value between 0 and 255} <value4>:={an integer value between 0 and 255}
QUERY SYNTAX	SYSTem:COMMunicate:LAN:IPADdress?
EXAMPLE	Set the IP address to 10.11.13.203: *SYST:COMM:LAN:IPAD "10.11.13.203"* Read IP address: *SYST:COMM:LAN:IPAD?* Return: *"10.11.13.203"*

3.13.10 Subnet mask command

DESCRIPTION	This command is used to set or obtain the subnet mask of the device.

COMMAND SYNTAX	SYSTem:COMMunicate:LAN:SMASk <parameter> <parameter>:=<value1>.<value2>.<value3>.<value4> <value1>:={an integer value between 0 and 255} <value2>:={an integer value between 0 and 255} <value3>:={an integer value between 0 and 255} <value4>:={an integer value between 0 and 255}
QUERY SYNTAX	SYSTem:COMMunicate:LAN:SMASk?
EXAMPLE	Set the subnet mask to 255.0.0.0: *SYST:COMM:LAN:SMAS "255.0.0.0"* Get subnet mask: *SYST:COMM:LAN:SMAS?* Return: *"255.0.0.0"*

3.13.11 Gateway command

DESCRIPTION	This command sets and gets the device's gateway.
COMMAND SYNTAX	SYSTem:COMMunicate:LAN:GATeway <parameter> <parameter>:=<value1>.<value2>.<value3>.<value4> <value1>:={an integer value between 1 and 223} <value2>:={an integer value between 0 and 255} <value3>:={an integer value between 0 and 255} <value4>:={an integer value between 0 and 255}
QUERY SYNTAX	SYSTem:COMMunicate:LAN:GATeway?
EXAMPLE	Set the gateway to 10.11.13.1: *SYSTem:COMMunicate:LAN:GATeway "10.11.13.1"* Get gateway: *SYSTem:COMMunicate:LAN:GATeway?* Return: *"10.11.13.1"*

3.13.12 GPIB command

DESCRIPTION	This command sets and gets the port number of Gpib.
COMMAND SYNTAX	SYSTem:COMMunicate:GPIB:ADDRess <num> <num>:= Gpib port number.

QUERY SYNTAX	SYSTem:COMMunicate:GPIB:ADDRess?
EXAMPLE	Set the Gpib port number to 19: *SYSTem:COMMunicate:GPIB:ADDRess 19* Get Gpib port number: *SYSTem:COMMunicate:GPIB:ADDRess?* Return: *19*

3.14 STore List command

DESCRIPTION	This command is used to read the name of the stored waveform data. If the storage unit is empty, this command will return the "EMPTY" field.
QUERY SYNTAX	STore List? STore List? <parameter> STore List? USER,<path>
RESPONSE FORMAT	<wave name>

parameter	value	describe
BUILDIN		Query the built-in waveform name and index number.
USER		Query the locally saved waveform name.

EXAMPLE

Read all arbitrary waveform data saved in the device (excluding data in the USB flash drive):

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

Read built-in waveform data stored in the device:

STL? BUILDIN

Return:

STL 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, Erfclnv, M113, Erflnv, 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 Read user-defined waveform name from device: *STL? USER* Return: *STLWVNM,sinc_8M,sinc_3000000,sinc_1664000, ramp_8M,sinc_2000000,sinc_50000,square_8M,sinc_5000,wave1, square_1M*

3.15 File management command

3.15.1 MMEMory:DELete

DESCRIPTION	This command is used to delete the specified file.
COMMAND SYNTAX	MMEMory:DELete <parameter> <parameter>:= File path (contains operation file name).
EXAMPLE	Delete the file with the path "Local/wave1.bin": *MMEMory:DELete "Local/wave1.bin"*

3.15.2 MMEMory:RDIRectory

DESCRIPTION	This command is used to delete the specified folder.
COMMAND SYNTAX	MMEMory:RDIRectory <parameter> <parameter>:= File path (contains operation file name).
EXAMPLE	Delete the folder named test under the folder path "Local/": *MMEMory:RDIRectory "Local/test"*

3.15.3 MMEMory:MDIRectory

DESCRIPTION	This command is used to create a new folder with the specified path.
COMMAND SYNTAX	MMEMory:MDIRectory <parameter> <parameter>:= File path (contains operation file name).
EXAMPLE	Create a new folder named test under the "Local" path: *MMEMory:MDIRectory "Local/test"*

3.15.4 MMEMory:CATalog

DESCRIPTION	This command is used to query all files and folders under the specified path or view files in the specified format.
QUERY SYNTAX	View all files and folders under the path: MMEMory:CATalog? <parameter> <parameter>:= folder path. MMEMory:CATalog:<parameter1>? <parameter2> <parameter1>:= STATe:XMLanguage.

	<parameter2>:= folder path.
EXAMPLE	View all files and folders under the "Local/" path: *MMEMory:CATalog? "Local"* View files with the ".arb" or ".ARB" suffix under the "Local/" path: *MMEMory:CATalog:DATA:ARBitrary? "Local"* Check the ".xml" or ".XML" suffix file under the "Local/" path: *MMEMory:CATalog:STATe:XMLanguage? "Local"*

3.15.5 MMEMory:COPY

DESCRIPTION	This command copies a file or folder.
COMMAND SYNTAX	MMEMory:COPY <parameter> <parameter>:= "Source file path", "Destination path".
EXAMPLE	Copy the file with the path "Local/test/123.bin" and paste it into "Local/123.bin": *MMEMory:COPY "Local/test/123.bin","Local/123.bin"* Copy the file/folder with the path "Local/src" and paste it into the "Local/copy/" folder: *MMEMory:COPY "Local/src","Local/copy"*

3.15.6 MMEMory:MOVE

DESCRIPTION	This command moves a file or folder to a new path.
COMMAND SYNTAX	MMEMory:MOVE <parameter> <parameter>:= "Source file path", "Destination path".
EXAMPLE	Move the file with the file path "Local/test/123.bin" to the path "Local/123.bin": *MMEMory:MOVE "Local/test/123.bin","Local/123.bin"* Move the folder path "Local/src" to the folder path "Local/copy/": *MMEMory:MOVE "Local/src", "Local/copy"*

3.15.7 MMEMory:SAVE:XML

DESCRIPTION	This command saves an xml configuration file to the default path or the specified path.

COMMAND SYNTAX	MMEMory:SAVE:XML <parameter> <parameter>:= Save path, including file name and suffix.
EXAMPLE	Save the test.xml file locally: *MMEMory:SAVE:XML "Local/test.xml" or* *MMEMory:SAVE:XML "test.xml"* Save the test.xml file to the USB flash drive: *MMEMory:SAVE:XML "U-disk0/test.xml"*

3.15.8 MMEMory:LOAD:XML

DESCRIPTION	This command loads an xml configuration file from the default path or the specified path.
COMMAND SYNTAX	MMEMory:LOAD:XML <parameter> <parameter>:= Save path, including file name and suffix.
EXAMPLE	Load test.xml file from local: *MMEMory:LOAD:XML "Local/test.xml" or* *MMEMory:LOAD:XML "test.xml"* Load test.xml file from USB flash drive: *MMEMory:LOAD:XML "U-disk0/test.xml"*

3.15.9 MMEMory:TRANsfer

DESCRIPTION	This command can send customized data to the specified bin file under the specified path.
COMMAND SYNTAX	MMEMory:TRANsfer <parameter>,#{data} <parameter>:= Save path, including file name and suffix. Note: This command is suitable for situations where the waveform data is small. When the waveform data is large, it is recommended to use the command in Chapter 3.6.5.3 to generate it.
EXAMPLE	Send a piece of data to the local wave1.bin: *MMEMory:TRANsfer "Local/wave1.bin",#14ABCD*

4 Waveform format

This chapter gives an explanation of the waveform file format used by the signal source. Through these instructions, you can learn how to customize and edit waveform files, and combine the commands listed in the previous chapter to control the signal source.

4.1 bin

The content of the bin file is binary, and the file content is the codeword value of each point (codeword range -32768~32767). Manual editing is not supported. When importing a file, the current amplitude, frequency, and offset information are maintained, and each codeword value is directly converted into a voltage output.

4.2 csv/dat

The content of csv and dat files is text. The CSV file is divided into two parts: header information segment and waveform data segment.

1. The csv header information segment contains the following four items. There can be more information items, but it must contain the following four items. More items will be ignored. Each item is on one line, ending with a newline character.

information item	describe
amp,value	amplitude of waveform
offset,value	Waveform offset
frequency,value	frequency of waveform
data length,value	The length of the waveform

2. Each group of data segment data occupies one line. One of the following four strings can be used as the starting identifier (the identifier occupies one line), placed before the formal data.
   Second,Volt
   xpos,value
   Time,Ampl
   Second,Value

An example of data in csv format is as follows:

1	data length	16384	
2	frequency	1000	
3	amp	2	
4	offset	0	
5	phase	0	
6			
7			
8			
9			
10			
11			
12			
13	xpos	value	
14	1	0	
15	2	0.000383	

The csv file removes the header information segment and only retains the data segment, which is the .dat format.

An example of dat format data is as follows:

```
Source,CH1
Second,Value
-1.0000000000E-04,-3.764706E-02
-9.9999800000E-05,-3.764706E-02
-9.9999600000E-05,-4.705882E-02
-9.9999400000E-05,-6.117647E-02
-9.9999200000E-05,-4.705882E-02
-9.9999000000E-05,-6.117647E-02
-9.9998800000E-05,-2.823529E-02
-9.9998600000E-05,-4.235294E-02
-9.9998400000E-05,-3.764706E-02
```

4.3 mat

1. mat file format description

The MAT file consists of a fixed-format 128-byte file header information (mat_head) and two data units.

a) The file header is represented by the following structure. Here we only need to pay attention to endian_indicator:

```
typedef struct
{
  char descriotive[116];
  char data_offset[8];
  uint16 version;      //Software version for exporting mat files
```

```
    char endian_indicator[2];          //The value is IM or MI, indicating big and small endian mode
}cfg_mat_header_t;
```

b) There is a data header information (data_head, 88 bytes) at the beginning of each data unit, which is used to describe the number of bytes occupied by the data unit, data type, data block name and other information.

The data block header is represented by the following structure:

```
typedef struct
{
    u32 data_type;      //The value must be CFG_MI_MATRIX
    u32 array_len;
    u32 array_flag_data_type;
    u32 array_flag_data_len;
    u32 array_flag_data_1;
    u32 array_flag_data_2;
    u32 dimensions_array_data_type;
    u32 dimensions_array_data_len;
    u32 dimensions_array_data_1;
    u32 dimensions_array_data_2;
    u32 array_name_data_type;
    u32 array_name_data_len;
    char array_name_data[32];       //The oscilloscope export file here is XX_time or XX_data
    u32 data_tag_data_type;             //The type of data in the following data area
    u32 data_tag_data_len;          //The size of the data in the following data area
}matlab_data_head_t;
```

Among them, XX_data_type represents the data type, which corresponds to the following enumeration value:

```
typedef enum
{
    CFG_MI_INT8=1,          //8 bit, signed
    CFG_MI_UINT8,           //8 bit, unsigned
    CFG_MI_INT16,           //16-bit, signed
    CFG_MI_UINT16,          //16-bit, unsigned
    CFG_MI_INT32,           //32-bit, signed
    CFG_MI_UINT32,          //32-bit, unsigned
    CFG_MI_SINGLE,          //IEEE 754 single format
```

```
    CFG_MI_RESERVED1,
    CFG_MI_DOUBLE,          //=9, IEEE 754 double format
    CFG_MI_RESERVED2,
    CFG_MI_RESERVED3,
    CFG_MI_INT64,           //=12, 64-bit, signed
    CFG_MI_UINT64,          //64-bit, unsigned
    CFG_MI_MATRIX,          //MATLAB array
    CFG_MI_COMPRESSED,      //Compressed Data
    CFG_MI_UTF8,            //Unicode UTF-8 Encoded Character Data
    CFG_MI_UTF16,           //Unicode UTF-16 Encoded Character Data
    CFG_MI_UTF32,           //Unicode UTF-32 Encoded Character Data
}cfg_mat_data_type_t;
```

2. The .mat files exported by matlab are in compressed format by default and need to be saved in non-compressed mode. And only supports files with two data segments. Time data needs to be double, waveform data needs to be single.

5 Waveform format

This chapter gives some programming examples. Through these examples, you can understand how to use VISA or sockets and combine the commands listed in the previous section to control the signal source. With the examples below, you can develop more applications.

5.1 VISA application example

5.1.1 VC++ example

Environment: Win7 32-bit system, Visual Studio

Description: Access the signal source through USBTMC and TCP/IP respectively, and send the "*IDN?" command on NI-VISA to query device information.

Step:

1. Open the Visual Studio software and create a new VC++ win32 console project.
2. Set up the project environment that calls the NI-VISA library. Two setting methods are given here, namely static and dynamic:

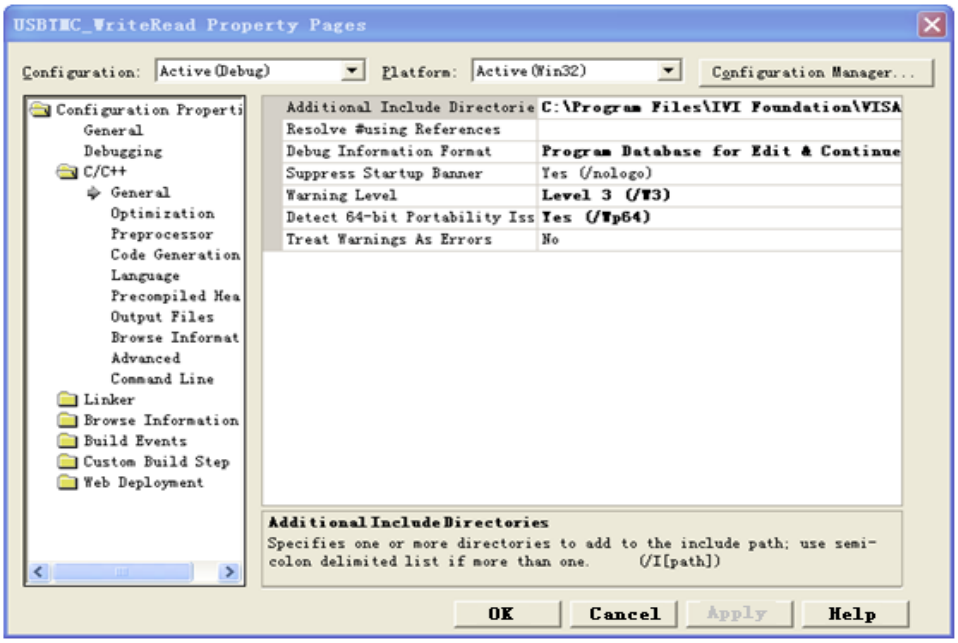
a) Static:

Find: visa.h, visatype.h, visa32.lib files in the NI-VISA installation path, copy them to the root path of the VC++ project and add them to the project. Add the following two lines of code to the projectname.cpp file:

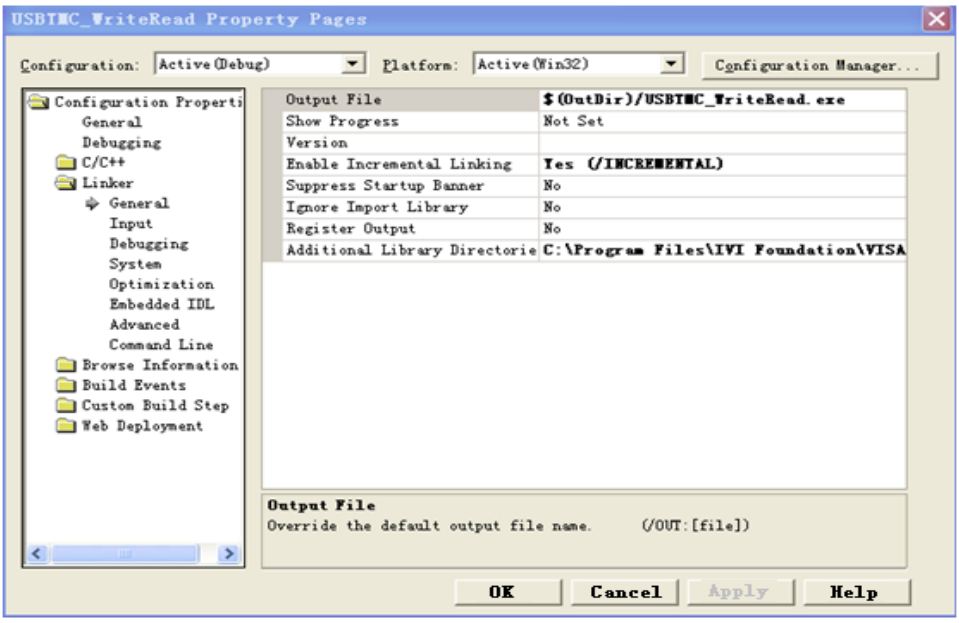
```
#include "visa.h"
#pragma comment(lib,"visa32.lib")
```

b) Dynamic:

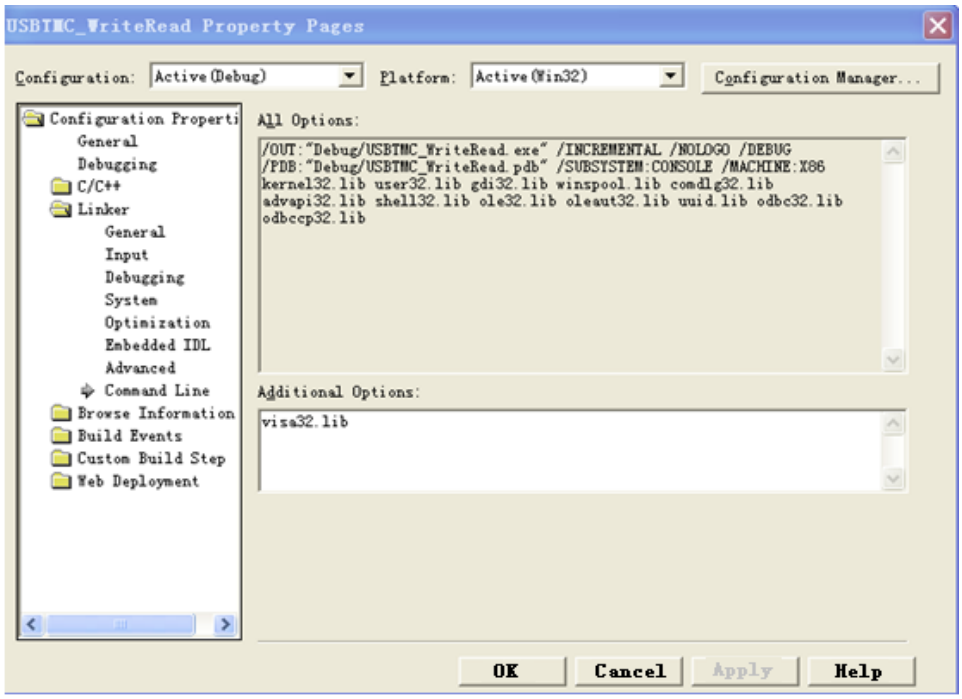
Click "project>>properties" and select "c/c++---General" on the left side of the properties dialog box. Set the value of the "Additional Include Directories" item to the installation path of NI-VISA (for example: C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the following figure:



Select "Linker---General" on the left side of the properties dialog box, and set the value of the "Additional Library Directories" item to the installation path of NI-VISA. (For example: C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown below:



Select "Linker---Command Line" on the left side of the properties dialog box, and set the value of the "Additional" item to visa32.lib, as shown in the following figure:



Add the visa.h file on the projectname.cpp file:

#include<visa.h>

3. Encoding:

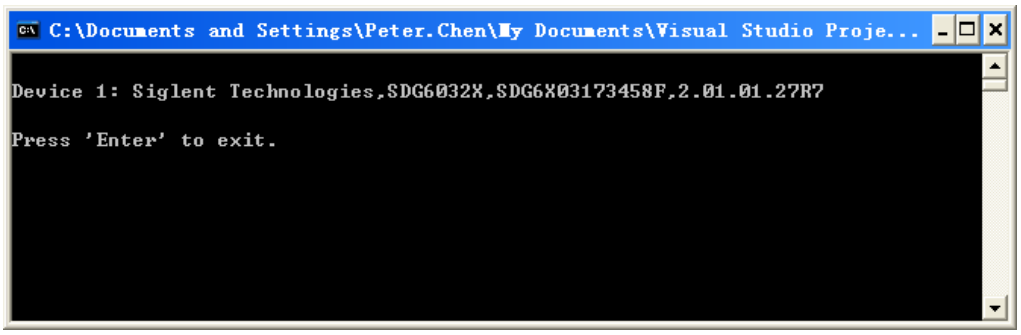
a) USBTMC:

```
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;
}
```

Running results:

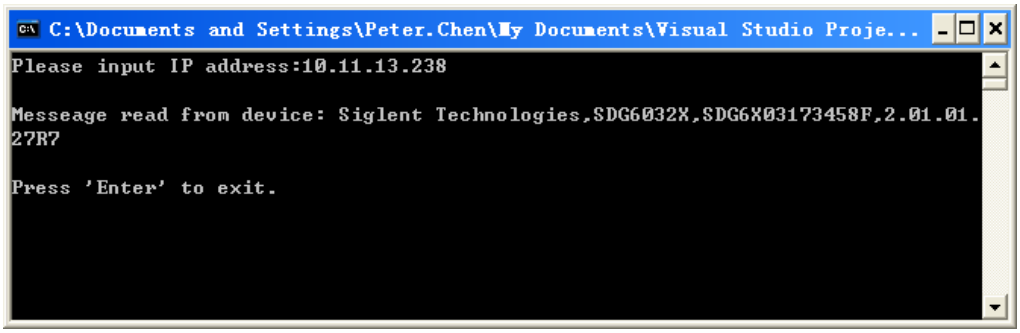


b) TCP/IP:

```
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;
}
```

Running results:



5.1.2 VB example

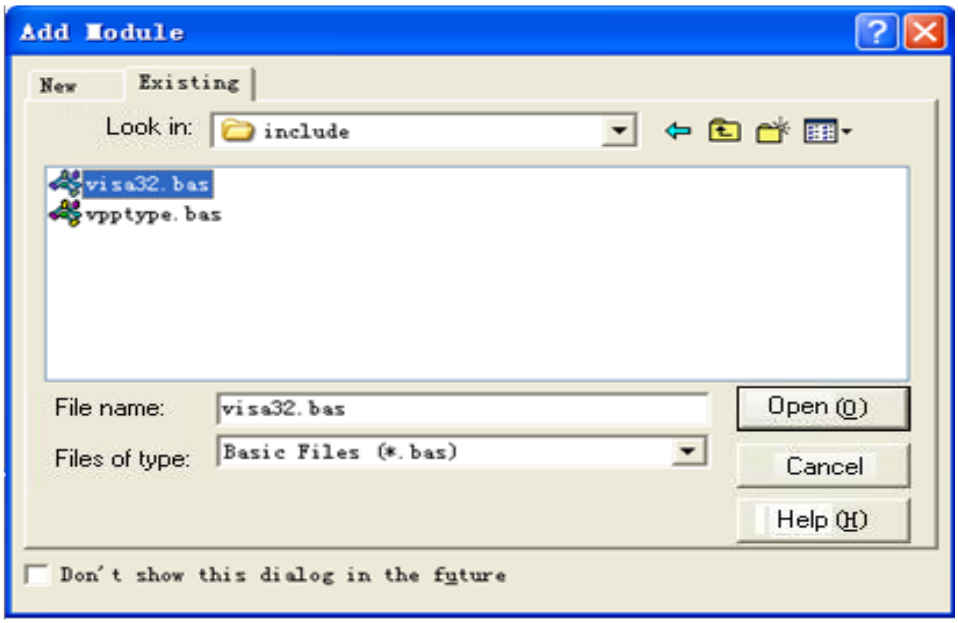
Environment: Windows7 32-bit system, Microsoft Visual Basic 6.0

Description: Access the signal source through USBTMC and TCP/IP respectively, and send the "*IDN?" command on NI-VISA to query device information.

Step:

1. Open the Visual Basic software and create a new standard application project.

Set the project environment for calling the NI-VISA library: click Existing tab of Project>>Add Existing Item, search for the visa32.bas file in the "include" folder under the NI-VISA installation path and add the file. As shown below:



2. Encoding:

a) USBTMC:

```
PrivateFunction Usbtmc_test() AsLong
' 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 AsLong
Dim instrsesn AsLong
Dim numInstrs AsLong
Dim findList AsLong
Dim retCount AsLong
Dim status AsLong
Dim instrResourceString AsString * VI_FIND_BUFLEN
Dim Buffer AsString * MAX_CNT
Dim i AsInteger

' 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
ExitFunction
EndIf

' 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) Then
        resultTxt.Text = "An error occurred while finding resources."
        viClose(defaultRM)
        Usbtmc_test = status
ExitFunction
EndIf

' 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)
    EndIf
                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
    EndIf

    ' 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 device'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
    EndIf

    ' 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
    EndIf
                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
EndFunction
```

b) TCP/IP:

```
PrivateFunction TCP_IP_Test(ByVal ip AsString) AsLong
Dim outputBuffer AsString * VI_FIND_BUFLEN
Dim defaultRM AsLong
```

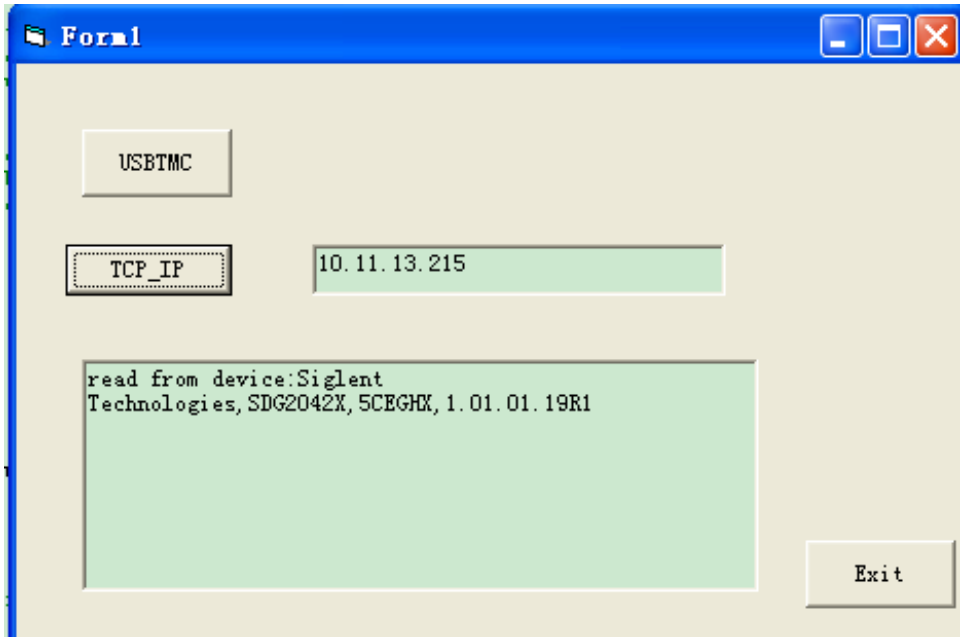
```
    Dim instrsesn AsLong
    Dim status AsLong
    Dim count AsLong

    ' 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
    ExitFunction
    EndIf

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

        status = viWrite(instrsesn, "*IDN?", 5, count)
    If (status < VI_SUCCESS) Then
            resultTxt.Text = "Error writing to the device."
    EndIf
        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
    EndIf
        status = viClose(instrsesn)
        status = viClose(defaultRM)
        TCP_IP_Test = 0
  EndFunction
```

c) Button control code:

```
    PrivateSub exitBtn_Click()
    End
    EndSub
    PrivateSub tcpipBtn_Click()
    Dim stat AsLong
```

```
        stat = TCP_IP_Test(ipTxt.Text)
    If (stat < VI_SUCCESS) Then
            resultTxt.Text = Hex(stat)
    EndIf
    EndSub
    PrivateSub usbBtn_Click()
    Dim stat AsLong
        stat = Usbtmc_test
    If (stat < VI_SUCCESS) Then
            resultTxt.Text = Hex(stat)
    EndIf
    EndSub
```

Running results:



5.1.3 MATLAB example

Environment: Windows 7 32-bit system, MATLAB R2013a

Description: Access the signal source through USBTMC and TCP/IP respectively, and send the "*IDN?" command on NI-VISA to query device information.

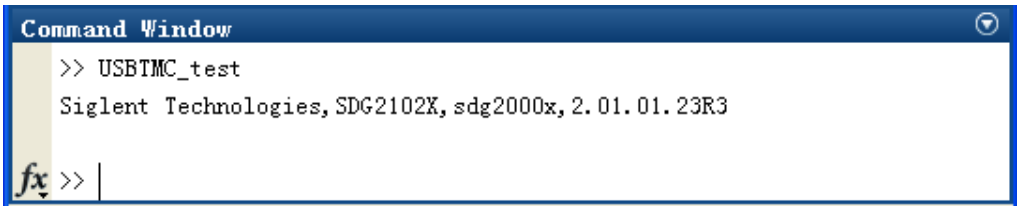
Step:

1. Open the MATLAB software and modify the current directory. In this example, 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. Encoding:

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::sdg2000x::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
```

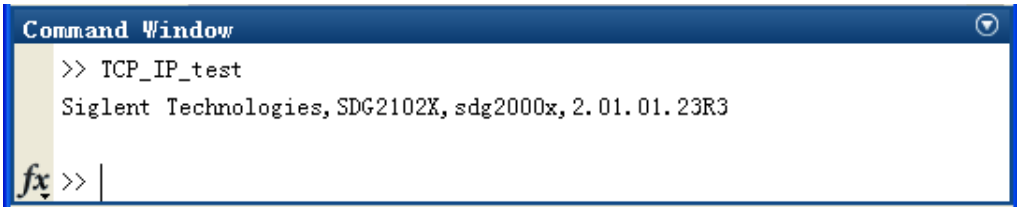
Running results:



b) TCP/IP:

```
function TCP_IP_test()
% This code demonstrates sending synchronous read & write commands
% to an 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
```

Running results:



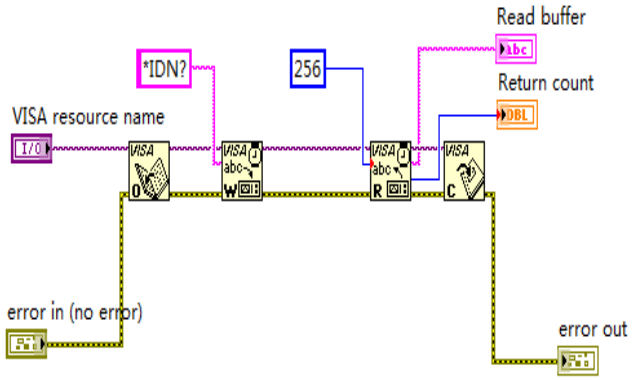
5.1.4 LabVIEW example

Environment: Windows 7 32-bit system, LabVIEW 2011

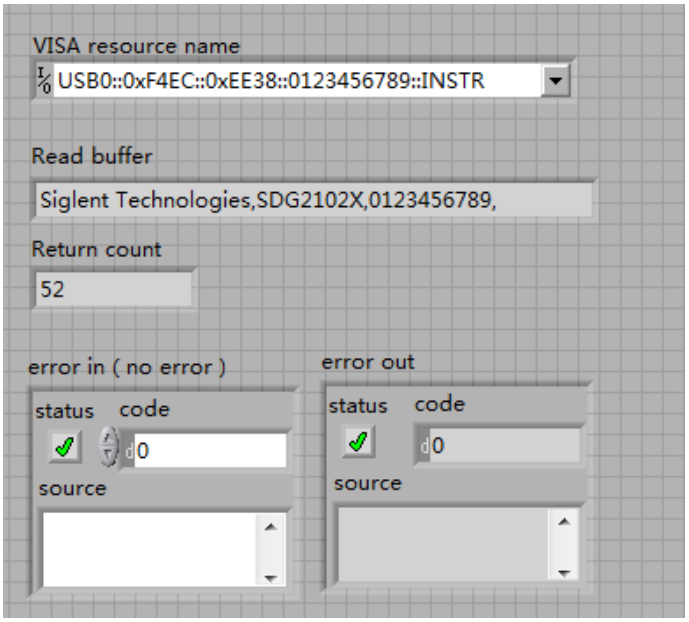
Description: Access the signal source through USBTMC and TCP/IP respectively, and send the "*IDN?" command on NI-VISA to query device information.

Step:

1. Open LabVIEW software and create a VI file.
2. Add controls. Right-click the front panel interface, select and add the VISA resource name, error input, error output, and partial indicators from the control column.
3. Open the block diagram interface. Right-click the VISA resource name, and select and add the following functions in the VISA Palette in the pop-up menu: VISA Write, VISA Read, VISA Open, and VISA Close.
4. Connect them as shown below:

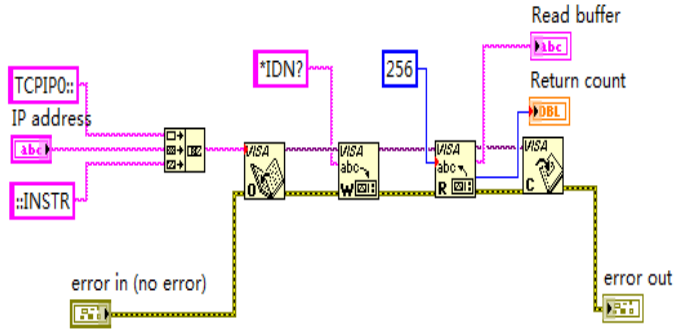


5. Select the device resource from the VISA resource name list and run the program.

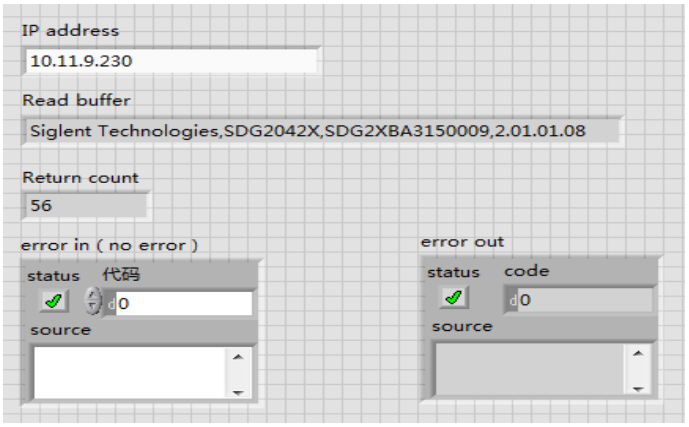


In this example, the VI opens a VISA session to a USBTMC device and writes *IDN? to the device. command, and the response value read back. When all communication is complete, the VI closes the VISA session.

6. Communicating with the device via TCP/IP is similar to USBTMC. But you need to set the VISA write function and VISA read function to synchronous I/O. LabVIEW is set to asynchronous IO by default. Right-click the node and select "Synchronous I/O Mod>>Synchronous" from the shortcut menu to write or read data synchronously.
7. Connect them as shown below:



8. Enter the IP address and run the program.



5.1.5 Python3 example1

Environment：Python3.6.5，PyVISA 1.9

Description: The number of waveform points created in the example. This value represents the codeword for each waveform point. Used to determine the level of the waveform point (the code word range of each waveform point is -32767~32767). Suitable for scenarios with a small number of waveform points. See Chapter 3.6.5.2 for details.

```
#!/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::SDG3000XABC002::INSTR")
data = [32767, 32767, 32767, 32767, 32767, 32767, 16384,16384,32767, 32767, 32767, 32767, 32767, 32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -16384,-16384,-32767, -32767, -32767, -32767, -32767]

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

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)
# By reading the data of the bin file, the existing waveform can be written to the device.
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()
```

5.1.6 Python3 example2

Environment: Python3.6.5, PyVISA 1.9

Description: The number of waveform points created in the example. This value represents the codeword for each waveform point. Used to determine the level of the waveform point (the code word range of each waveform point is -32767~32767). Suitable for scenarios with a small number of waveform points. See Chapter 3.6.5.3 for details.

```
#!/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::SDG3000XABC002::INSTR")
data = [32767, 32767, 32767, 32767, 32767, 32767, 16384,16384,32767, 32767, 32767, 32767, 32767, 32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -16384,-16384,-32767, -32767, -32767, -32767, -32767]

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

byte_data1 = b''.join(int_to_two_byte_small_endian(num) for num in data[:8])
print('write bytes:', len(byte_data1))
print(byte_data1)
byte_data2 = b''.join(int_to_two_byte_small_endian(num) for num in data[8:16])
print('write bytes:', len(byte_data2))
print(byte_data2)
byte_data3 = b''.join(int_to_two_byte_small_endian(num) for num in data[16:])
print('write bytes:', len(byte_data3))
print(byte_data3)
cmd1 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",FRQ,3500,AMP,1.5,OFST,0.01,BEGIN,WAVEDATA,', 'utf-8') + byte_data1
cmd2 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",WAVEDATA,', 'utf-8') + byte_data2
cmd3 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",END,WAVEDATA,', 'utf-8') + byte_data3
sdg.write_raw(cmd1)
sdg.write_raw(cmd2)
sdg.write_raw(cmd3)
```

```
# By reading the data of the bin file, the existing waveform can be written to the device.
f = open(r"C:\Users\Administrator\Desktop\wave1.bin", "rb")
data1 = f.read()
cmd1 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",FRQ,3500,AMP,1.5,OFST,0.01,BEGIN,WAVEDATA,', 'utf-8') + data1[:8]
cmd2 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",WAVEDATA,', 'utf-8') + data1[8:16]
cmd3 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",END,WAVEDATA,', 'utf-8') + data1[16:]
sdg.write_raw(cmd1)
sdg.write_raw(cmd2)
sdg.write_raw(cmd3)
f.close()
```

5.2 Sockets application example

5.2.1 Python example

Python has a low-level network module for accessing socket interfaces. Python scripts written based on sockets can be used to perform various test and measurement tasks.

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

Description: Open a socket, send a query operation and loop it 10 times before closing the socket.

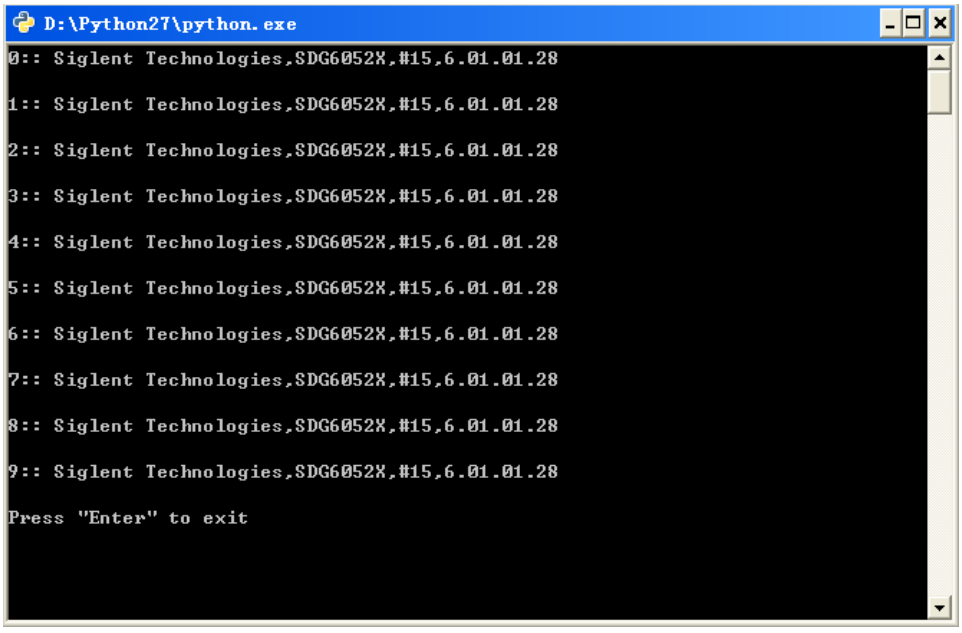
Note: The SCPI command string in the program must end with the "\n" character (newline).

Here is the code for the script:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
#------------------------------------------------------------------------------
# The short script is a example that open 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()
```

Running results:



6 Index

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 America, Inc
6557 Cochran Rd Solon, Ohio 44139
Tel: 440-398-5800
Toll Free: 877-515-5551
Fax: 440-399-1211
Email: info@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