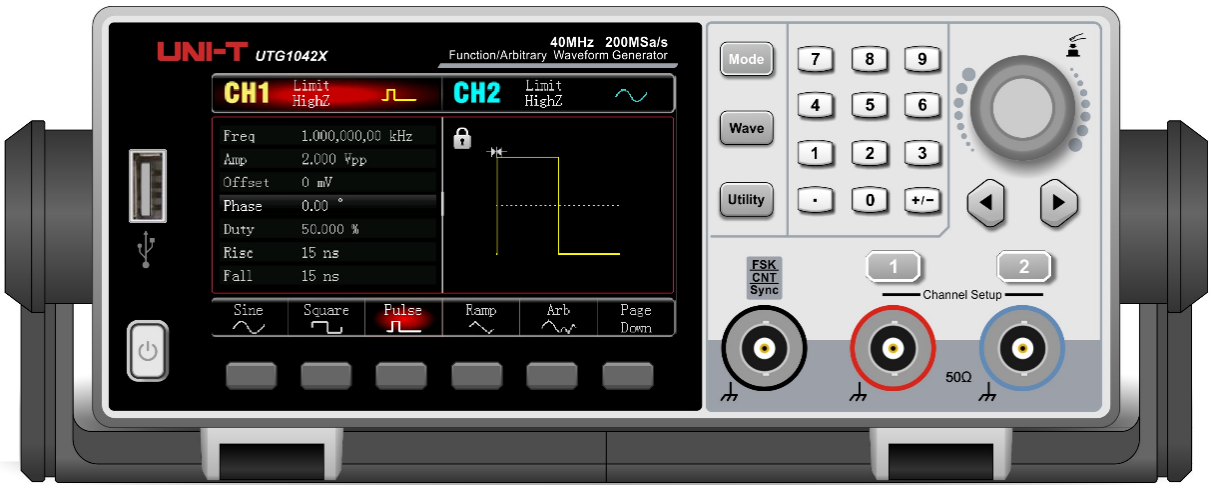
UNI-T®

Instruments.uni-trend.com



Programming Manual

UTG1000X Series Function/Arbitrary Waveform Generator

Warranty and Statement

Copyright

Brand Information

UNI-T is the registered trademark of Uni-Trend Technology (China) Co., Ltd.

Software Version

3.07.000

Software upgrade may have some change and add more function, please subscribe UNI-T website to get the new version or contact UNI-T.

Statement

- UNI-T products are protected by patents (including obtained and pending) in China and other countries and regions.
- UNI-T reserves the right to change specifications and prices.
- The information provided in this manual supersedes all previous publications.
- The information provided in this manual is subject to change without notice.
- UNI-T shall not be liable for any errors that may be contained in this manual. For any incidental or consequential damages arising out of the use or the information and deductive functions provided in this manual.
- No part of this manual shall be photocopied, reproduced or adapted without the prior written permission of UNI-T.

Product Certification

UNI-T has certified that the product conforms to China national product standard and industry product standard as well as ISO9001:2008 standard and ISO14001:2004 standard. UNI-T will go further to certificate product to meet the standard of other member of the international standards organization.

Contact Us

If you have any question or problem, please can contact UNI-T.

Website: https://www.uni-trend.com

SCPI

SCPI（Standard Commands for Programmable Instruments）is a standardized instrument programming language that builds on existing standards IEEE 488.1 and IEEE 488.2 and follows the floating point rules of IEEE 754 standard, ISO 646 message exchange 7-bit encoding notation (equivalent to ASCII programming) and many other standards.

This section introduces the format, symbols, parameters, and abbreviations of the SCPI command.

Instruction Format

The SCPI command is a tree-like hierarchy consisting of multiple subsystems, each subsystem consisting of a root keyword and one or more hierarchical key words. **The command line usually begins with a colon ":"; Keywords are separated by the colon ":", followed by optional parameter settings. The command keyword is separated by spaces from the first parameter. The command string must end with a newline <NL> character. Add the question mark "?" after the command line. It is usually indicated that this feature is being queried.**

Symbol Description

The following four symbols are not part of the SCPI command. They cannot be sent with the command, but they are commonly used for supplementary specification.

- Braces { }

  It usually contains multiple optional parameter, one of which must be selected when send a command.

  For example, the command :DISPlay:GRID:MODE　{FULL | GRID | CROSS | NONE}

- Vertical Bar |

  It is used to separate multiple parameters, one of which must be selected when send a command.

  For example, the command :DISPlay:GRID:MODE　{FULL | GRID | CROSS | NONE}

- Square Brackets [ ]

  The contents in square brackets (command keywords) can be omitted. If the parameter is omitted, the instrument will set the parameter to the default value.

  For example, the command :MEASure:NDUTy? [<source>], [<source>] represents the current channel.

- Angle Braces < >

  The parameter enclosed in the angle brackets must be replaced by an effective value.

For example, use the command DISPlay:GRID:BRIGhtness 30 to send the command DISPlay:GRID:BRIGhtness <count>

Parameter Description

The parameter in this manual can be divided into five types: Boolean, Integer, Real, Discrete and ASCII.

- **Boolean**
  The available values for the parameter is "ON"（1） or "OFF"（0）.
  For example, :SYSTem:LOCK {{1|ON}|{0|OFF}}.

- **Integer**
  Unless otherwise specified, the parameter can be any integer within the effective value range.
  Notice: Do not set the parameter to a decimal or in scientific notation, otherwise, errors will occur.
  For example, :DISPlay:GRID:BRIGhtness <count>, <count> can take integer form 0-100.

- **Real**
  Unless otherwise specified, the parameter can be any value within the effective value range.
  For example, for CH1, <offset> in the command CHANnel1:OFFSet <offset> can take value as real.

- **Discrete**
  The parameter can only take a few specified number or characters.
  For example, the parameter in the command :DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE}, it can only be FULL, GRID, CROSS or NONE.

- **ASCII**
  Character string parameter can contain all ASCII sets. Character string must begin and end with paired quotes; it can use single or double quotation marks. The quotation and delimiter can also be part of a string by typing it twice and not adding any characters.
  For example, set IP SYST:COMM:LAN:IPAD "192.168.1.10"

Shorthand Rule

All the commands are case-insensitive. The commands can be all input in uppercase letters or in lowercase letters. For abbreviations, it should enter all the uppercase letters that exist in the command syntax.

Data Return

Data return is divided into single data and batch data. The single data return is the corresponding parameter type, in which the real return type is presents by the scientific notation method. The part before e retains three figure behind the decimal point, and the e part retains three figure; the batch return must be obey IEEE 488.2# string data format, '#'+ the length of character bits [fixed to one character] + ASCII valid value+ valid data+ end string ['\n']

For example, #3123xxxxxxxxxxxxxxxxxxx\n represents 123 strings batch data return format, '3' represents "123" occupies three character bits.

SCPI Explanation

IEEE488.2 General Command

*IDN?

- Syntax
  *IDN?
- Description
  Query for manufacture name, product model, product serial number and software version number.
- Return Format
  The query returns manufacture name, product model, and product serial number. The software version is separated by dot mark.
- For Example
  UNI-T Technologies, UTG1000X, 000000001, 00.00.01

*RST

- Syntax
  *RST
- Description
  To restore the instrument to its factory default settings, clear all the error message and send and receive queue buffers.

SYSTem Command

It is used for the basic operation of the signal source, including voice selection and system data setting.

:SYSTem:CONFigure

- Syntax
  :SYSTem:CONFigure <file>
  :SYSTem:CONFigure?
- Description
  Write/read the configuration file. Send the instruction at first, and then send the configuration file data to the signal source.
  <file> represents the configuration file.
- Return Format
  The query returns the currently configuration file data of the signal source.
- For Example
  :SYSTem:CONFigure
  Write the configuration file data into the signal source and load it.
  :SYSTem:CONFigure?
  The query returns the currently configuration file data of the signal source. It's binary system.

:SYSTem:PHASe:MODe

- Syntax

  :SYSTem:PHASe:MODe {INDependent | SYNChronization}

  :SYSTem:PHASe:MODe?
- Description

  The phase mode of the interchannel. If the mode is sync, it represents start phase of the two channels keeps synchronized. Otherwise, the phase mode is independent.
- Return Format

  The query returns the phase mode between in channels.
- For Example

  :SYSTem:PHASe:MODe INDependent

  Set the phase mode of the interchannel to INDependent.

  :SYSTem:PHASe:MODe? The query returns INDependent.

:SYSTem:LANGuage

- Syntax

  :SYSTem:LANGuage {ENGLish|CHINese}

  :SYSTem:LANGuage?
- Description

  The system language.
- Return Format

  The query returns the system language.
- For Example

  :SYSTem:LANGuage ENGLish Set the system language to ENGLish.

  :SYSTem:LANGuage? The query returns ENGLish.

:SYSTem:BEEP

- Syntax

  :SYSTem:BEEP {{1 | ON} | {0 | OFF}}

  :SYSTem:BEEP?
- Description

  The switch of beeper.
- Return Format

  The query returns the switch status of beeper.
- For Example

  :SYSTem:BEEP ON Turn on beeper.

  :SYSTem:BEEP? The query returns 1.

:SYSTem:NUMBer:FORMat

- Syntax

  :SYSTem:NUMBer:FORMat {COMMa|SPACe|NONe}

  :SYSTem:NUMBer:FORMat?

- **Description**

  A separator of number forma for the system.

- **Return Format**

  The query returns separator in system number format.

- **For Example**

  :SYSTem:NUMBer:FORMat NONe Set the number format of the system to NONe.

  :SYSTem:NUMBer:FORMat? The query returns NONe.

:SYSTem:BRIGhtness

- **Syntax**

  :SYSTem:BRIGhtness {10|30|50|70|90|100}

  :SYSTem:BRIGhtness?

- **Description**

  The backlight brightness level of the system.

- **Return Format**

  The query returns the backlight brightness level of the system.

- **For Example**

  :SYSTem:BRIGhtness 30 Set the backlight brightness level to 30%.

  :SYSTem:BRIGhtness? The query returns 30.

:SYSTem:SLEEP:TIMe

- **Syntax**

  : SYSTem:SLEEP:TIMe {CLOSe| 1MIN | 5MIN | 15MIN | 30MIN | 60MIN}

  :SYSTem:SLEEP:TIMe?

- **Description**

  The sleep time of the system, the unit is minute.

- **Return Format**

  The query returns the sleep time.

- **For Example**

  :SYSTem:SLEEP:TIMe 5MIN Set the sleep time to 5MIN.

  :SYSTem:SLEEP:TIMe? The query returns 5MIN.

:SYSTem:CYMometer

- **Syntax**

  :SYSTem:CYMometer {{1 | ON} | {0 | OFF}}

  :SYSTem:CYMometer?

- **Description**

  Control the switch status of system's frequency meter.

  Notice: If this function is enabled, channel's sync output will be turned off.

- **Return Format**

  The query returns the switch status of system's frequency meter. 0 represents OFF, 1 represents ON.

- **For Example**

  :SYSTem:CYMometer ON Turn on system's frequency meter.

:SYSTem:CYMometer? The query returns 1.

:SYSTem:CYMometer:FREQuency?

- Syntax
  :SYSTem:CYMometer:FREQuency?
- Description
  Acquiring the currently measured frequency of the frequency meter.
- Return Format
  The query returns the currently measured frequency of the frequency meter in scientific notation. The unit is Hz.
- For Example
  :SYSTem:CYMometer:FREQuency? The query returns 2.000000e +3.

:SYSTem:CYMometer:PERiod?

- Syntax
  :SYSTem:CYMometer:PERiod?
- Description
  Acquiring the currently measured period of the frequency meter.
- Return Format
  The query returns the currently measured period of the frequency meter in scientific notation. The unit is S.
- For Example
  :SYSTem:CYMometer:PERiod? The query returns 2.000000e -3.

:SYSTem:CYMometer:DUTY?

- Syntax
  :SYSTem:CYMometer:DUTY?
- Description
  Acquiring the currently measured duty cycle of the frequency meter. Return data in scientific notation.
- Return Format
  The query returns the currently measured duty cycle of the frequency meter. The unit is %.
- For Example
  :SYSTem:CYMometer:DUTY?
  The query returns 2.000000e+01, it represents duty cycle is 20%.

CHANnel Command

It used to set the channel function of the signal source.

:CHANnel<n>:MODE

- Syntax

  :CHANnel<n>:MODE {CONTinue|MODulation|SWEep|BURSt }

  :CHANnel<n>:MODE?

- Description

  Set the signal mode for the specified channel, which is CONTinue, MODulation, SWEep or BURSt.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the signal mode of the specified channel.

- For Example

  :CHANnel1:MODE MODulation Set signal mode of CH1 to MODulation.

  :CHANnel1:MODE? The query returns MODulation.

:CHANnel<n>:OUTPut

- Syntax

  :CHANnel<n>:OUTPut {{1|ON}|{0|OFF}}

  :CHANnel<n>:OUTPut?

- Description

  Turn on/off the channel output switch of the specified channel.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the output status of the specified channel. 0 represents OFF, 1 represents ON.

- For Example

  :CHANnel1:OUTPut ON Turn on CH1 output.

  :CHANnel1:OUTPut? The query returns 1.

:CHANnel<n>:PA:OUTPut

- Syntax

  :CHANnel<n>:PA:OUTPut {{1|ON}|{0|OFF}}

  :CHANnel<n>:PA:OUTPut?

- Description

  Turn on/off power amplifier output.

  <n>：Channel number, n takes value 2.

  Notice：This function can only enabled when it has power amplifier module. Normal CH2 and power amplifier output are mutually exclusive, and normal CH2 cannot be used at this time.

- Return Format

  The query returns the status of power amplifier output. 0 represents OFF, 1 represents ON.

- For Example

  :CHANnel2:PA:OUTPut ON Turn on power amplifier output.

:CHANnel2:PA:OUTPut? The query returns 1.

:CHANnel<n>:INVersion

- Syntax

  :CHANnel<n>:INVersion {{1|ON}|{0|OFF}}

  :CHANnel<n>:INVersion?

- Description

  Turn on/off the inversion of the specified channel.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the inversion status of the specified channel. 0 represents OFF, 1 represents ON.

- For Example

  :CHANnel1:INVersion ON Turn on reverse output of CH1.

  :CHANnel1:INVersion? The query returns 1.

:CHANnel<n>:OUTPut:SYNC

- Syntax

  :CHANnel<n>:OUTPut:SYNC {{1|ON}|{0|OFF}}

  :CHANnel<n>:OUTPut:SYNC?

- Description

  Set the sync output status for the specified channel.

  Notice: If there is only one synchronous output interface, and only one channel can be opened for synchronous output.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the sync output status of the specified channel. 0 represents OFF, 1 represents ON.

- For Example

  :CHANnel1:OUTPut:SYNC ON Turn on sync output of CH1.

  :CHANnel1:OUTPut:SYNC? The query returns 1.

:CHANnel<n>:LIMit:ENABle

- Syntax

  :CHANnel<n>:LIMit:ENABle {{1|ON}|{0|OFF}}

  :CHANnel<n>:LIMit:ENABle?

- Description

  Set the amplitude limit switch for the specified channel.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the amplitude limit status of the specified channel.

- For Example

  :CHANnel1:LIMit:ENABle ON Turn on amplitude limit of CH1.

  :CHANnel1:LIMit:ENABle? The query returns 1.

:CHANnel<n>:LIMit:LOWer

- **Syntax**

  :CHANnel<n>:LIMit:LOWer {<voltage>}

  :CHANnel<n>:LIMit:LOWer?

- **Description**

  Set the lower limit of amplitude for the specified channel.

  <voltage> represents voltage, the unit is the specified unit of the current channel.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the lower limit of amplitude of the specified channel in scientific notation.

- **For Example**

  :CHANnel1:LIMit:LOWer 2 Set the lower limit of amplitude for CH1 to 2V.

  :CHANnel1:LIMit:LOWer? The query returns 2e+0.

:CHANnel<n>:LIMit:UPPer

- **Syntax**

  :CHANnel<n>:LIMit:UPPer {<voltage>}

  :CHANnel<n>:LIMit:UPPer?

- **Description**

  Set the upper limit of amplitude for the specified channel.

  <voltage> represents voltage, the unit is the specified unit of the current channel.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the upper limit of amplitude of the specified channel in scientific notation.

- **For Example**

  :CHANnel1:LIMit:UPPer 2 Set the upper limit of amplitude for CH1 to 2V.

  :CHANnel1:LIMit:UPPer? The query returns 2e+0.

:CHANnel<n>:AMPLitude:UNIT

- **Syntax**

  :CHANnel<n>:AMPLitude:UNIT { VPP|VRMS|DBM}

  :CHANnel<n>:AMPLitude:UNIT?

- **Description**

  Set the output amplitude unit for the specified channel. If the system is in high/low level mode, this setting is invalid.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output amplitude unit of the specified channel.

- **For Example**

  :CHANnel1:AMPLitude:UNIT VPP Set the output amplitude unit for CH1 to VPP.

  :CHANnel1:AMPLitude:UNIT? The query returns VPP.

:CHANnel<n>:LOAD

- **Syntax**

  :CHANnel<n>:LOAD {<resistance>}

  :CHANnel<n>:LOAD?

- **Description**

  Set the output load for the specified channel.

  <resistance> represents load resistance value, the unit isΩ.

  <n>: Channel number, n takes value 1, 2.

  Notice: The range of resistance value is 1~10000, 10000 is corresponding to high resistance.

- **Return Format**

  The query returns load resistance value of the specified channel in scientific notation.

- **For Example**

  :CHANnel1:LOAD 50 Set the output load for CH1 to 50Ω.

  :CHANnel1:LOAD? The query returns 5.000000e+01.

:CHANnel<n>:SELect

- **Syntax**

  :CHANnel<n>:SELect

  :CHANnel<n>:SELect?

- **Description**

  To select the channel.

  <n>: {1|2}, it respectively represents {CH1|CH2}.

- **Return Format**

  The query returns 1 or 0, it respectively represents ON or OFF.

- **For Example**

  :CHAN1:SELect Select CH1.

  :CHAN1:SELect?

  The query returns 1, it represents the channel is selected.

Continuous

:CHANnel<n>:BASE:WAVe

- **Syntax**

  :CHANnel<n>:BASE:WAVe { SINe | SQUare | PULSe | RAMP | ARB | NOISe | DC }

  :CHANnel<n>:BASE:WAVe?

- **Description**

  Set the basic wave for the specified channel, which is sine wave, square wave, impulse wave, triangle wave, arbitrary wave, noise wave and DC.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the basic wave of the specified channel.

- **For Example**

  :CHANnel1:BASE:WAVe SINe Set the basic wave of CH1 to sine wave.

  :CHANnel1:BASE:WAVe? The query returns SINe.

:CHANnel<n>:BASE:FREQuency

- **Syntax**

  :CHANnel<n>:BASE:FREQuency {<freq>}

  :CHANnel<n>:BASE:FREQuency?

- **Description**

  Set the output frequency for the specified channel.

  <freq> represents frequency value, the unit is Hz. (1e-6Hz ~ the allowed maximum frequency of current wave).

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output frequency of the specified channel in scientific notation.

- **For Example**

  :CHANnel1:BASE:FREQuency 2000 Set the output frequency of CH1 to 2KHz.

  :CHANnel1:BASE:FREQuency? The query returns 2e+3.

:CHANnel<n>:BASE:PERiod

- **Syntax**

  :CHANnel<n>:BASE:PERiod {<period>}

  :CHANnel<n>:BASE:PERiod?

- **Description**

  Set the output period for the specified channel.

  <period> represents period, the unit is S.

  If it is sine wave, the range is from currently allowed maximum time to 1e6s.

<n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the upper limit of amplitude of the specified channel in scientific notation.

- **For Example**

:CHANnel1:BASE:PERiod 0.002	Set the output period of CH1 to 2ms.
:CHANnel1:BASE:PERiod?	The query returns 2e-3.

:CHANnel<n>:BASE:PHASe

- **Syntax**

  :CHANnel<n>:BASE:PHASe { <phase>}

  :CHANnel<n>:BASE:PHAse?

- **Description**

  Set the output phase for the specified channel.

  <phase> represents phase, the unit is°. The range is -360~360.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output phase of the specified channel.

- **For Example**

:CHANnel1:BASE:PHAse 20	Set the output phase of CH1 to 20°.
:CHANnel1:BASE:PHAse?	The query returns 20.

:CHANnel<n>:BASE:AMPLitude

- **Syntax**

  :CHANnel<n>:BASE:AMPLitude {<amp>}

  :CHANnel<n>:BASE:AMPLitude?

- **Description**

  Set the output amplitude for the specified channel.

  <amp> represents voltage, the unit is the specified unit of the current channel. 1mVpp ~ the maximum output value under the current load.

  If the current unit is VPP, the maximum value under the current load = the current load *20/(50+ the current load)

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output amplitude of the specified channel in scientific notation.

- **For Example**

:CHANnel1:BASE:AMPLitude 2	Set the output amplitude of CH1 to 2V.
:CHANnel1:BASE:AMPLitude?	The query returns 2e+0.

:CHANnel<n>:BASE:OFFSet

- **Syntax**

:CHANnel<n>:BASE:OFFSet {<voltage>}

:CHANnel<n>:BASE:OFFSet?

- Description

Set the output DC offset for the specified channel.

<voltage> represents voltage, the unit is V. The range is from 0 to the maximum DC under the current load.

The maximum DC under the current load = the current load *10/ (50+ the current load) – the minimum value under the current load. DC mode under /2

AC minimum value is 2mVpp, DC mode takes 0;

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the output DC offset of the specified channel in scientific notation.

- For Example

:CHANnel1:BASE:OFFSet 2	Set the output DC offset of CH1 to 2V.
:CHANnel1:BASE:OFFSet?	The query returns 2e+0.

:CHANnel<n>:BASE:HIGH

- Syntax

:CHANnel<n>:BASE:HIGH {<voltage>}

:CHANnel<n>:BASE:HIGH?

- Description

Set the high level of signal output for the specified channel.

<voltage> represents voltage, the unit is the specified unit of the current channel.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the high level of signal output of the specified channel in scientific notation.

- For Example

:CHANnel1:BASE:HIGH 2	Set the high level of signal output for CH1 to 2V.
:CHANnel1:BASE:HIGH?	The query returns 2e+0.

:CHANnel<n>:BASE:LOW

- Syntax

:CHANnel<n>:BASE:LOW {<voltage>}

:CHANnel<n>:BASE:LOW?

- Description

Set the lower level of signal output for the specified channel.

<voltage> represents voltage, the unit is the specified unit of the current channel.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the lower level of signal output of the specified channel in scientific notation.

- For Example

:CHANnel1:BASE:LOW 2	Set the lower level of signal output for CH1 to 2V.
:CHANnel1:BASE:LOW?	The query returns 2e+0.

:CHANnel<n>:BASE:DUTY

- **Syntax**

  :CHANnel<n>:BASE:DUTY {<duty>}

  :CHANnel<n>:BASE:DUTY?

- **Description**

  Set the output duty cycle for the specified channel.

  <duty> represents duty cycle, the unit is %. The range is 0-100.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output duty cycle of the specified channel.

- **For Example**

  :CHANnel1:BASE:DUTY 20 Set the output duty cycle of CH1 to 20% .

  :CHANnel1:BASE:DUTY? The query returns 20.

:CHANnel<n>:BASE:ARB

- **Syntax**

  :CHANnel<n>:BASe:ARB <source>,<filename>

  :CHANnel<n>:BASe:ARB?

- **Description**

  Set the specified channel to load arbitrary wave data of a file under basic wave and arbitrary wave source.

  <n>: Channel number, n takes value 1, 2.

  <source>: {INTernal|EXTernal }, which is internal or external.

  <filename>: The filename of arbitrary waveform.

- **For Example**

  : CHANnel1:BASe:ARB INTernal, "AbsSine.bsv"

:CHANnel<n>:RAMP:SYMMetry

- **Syntax**

  :CHANnel<n>:RAMP:SYMMetry { < symmetry >}

  :CHANnel<n>:RAMP:SYMMetry?

- **Description**

  Set the output symmetry of ramp signal for the specified channel.

  < symmetry > represents symmetry, the unit is %. The range is 0-100.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the output symmetry of ramp signal for the specified channel in scientific notation.

- **For Example**

  :CHANnel1:RAMP:SYMMetry 20

  Set the output symmetry of ramp signal for CH1 to 20%.

  :CHANnel1:RAMP:SYMMetry? The query returns 2.000000e+01.

:CHANnel<n>:PULSe:RISe

- **Syntax**

  :CHANnel<n>:PULSe:RISe {<width>}

  :CHANnel<n>:PULSe:RISe?

- **Description**

  Set the rising edge pulse width of impulse wave for the specified channel.

  <width> represents pulse width, the unit is S.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the rising edge pulse width of impulse wave for the specified channel in scientific notation.

- **For Example**

  :CHANnel1:PULSe:RISe 0.002

  Set the rising edge pulse width of impulse wave for CH1 to 2ms.

  :CHANnel1:PULSe:RISe? The query returns 2e-3.

:CHANnel<n>:PULSe:FALL

- **Syntax**

  :CHANnel<n>:PULSe:FALL {<width>}

  :CHANnel<n>:PULSe:FALL?

- **Description**

  Set the falling edge pulse width of impulse wave for the specified channel.

  <width> represents pulse width, the unit is S.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the falling edge pulse width of impulse wave for the specified channel in scientific notation.

- **For Example**

  :CHANnel1:PULSe:FALL 0.002

  Set the falling edge pulse width of impulse wave for CH1 to 2ms.

  :CHANnel1:PULSe:FALL? The query returns 2e-3.

Modulation

:CHANnel<n>:MODulate:TYPe

- **Syntax**

  :CHANnel<n>:MODulate:TYPe <type>

  :CHANnel<n>:MODulate:TYPe?

- **Description**

  Set the modulation type for the specified channel.

  <type>: { AM|FM|PM|ASK|FSK|PSK|PWM }, which is AM, FM, PM, ASK, FSK, PSK and PWM.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the modulation type of the specified channel.

- **For Example**

  :CHANnel1:MODulate:TYPe AM Set CH1 signal to AM.

  :CHANnel1:MODulate:TYPe? The query returns AM.

:CHANnel<n>:MODulate:WAVe

- **Syntax**

  :CHANnel<n>:MODulate:WAVe { SINe|SQUare|UPRamp|DNRamp|ARB|NOISe }

  :CHANnel<n>:MODulate:WAVe?

- **Description**

  Set the modulation wave for the specified channel. There are sine wave, square wave, upper triangle wave, down triangle wave arbitrary wave, arbitrary wave and noise wave.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the modulation wave of the specified channel.

- **For Example**

  :CHANnel1:MODulate:WAVe SINe Set the modulation wave of CH1 to SINe.

  :CHANnel1:MODulate:WAVe? The query returns SINe.

:CHANnel<n>:MODulate:SOURce

- **Syntax**

  :CHANnel<n>:MODulate:SOURce { INTernal|EXTernal }

  :CHANnel<n>:MODulate:SOURce?

- **Description**

  Set the modulation source for the specified channel, which is internal and external.

  <n>: Channel number, n takes value 1, 2.

- **Return Format**

  The query returns the modulation source of the specified channel in scientific notation.

- For Example

  :CHANnel1:MODulate:SOURce INTernal

  Set the modulation source of CH1 to INTernal.

  :CHANnel1:MODulate:SOURce? The query returns INTernal.

:CHANnel<n>:MODulate:FREQuency

- Syntax

  :CHANnel<n>:MODulate:FREQuency {<freq>}

  :CHANnel<n>:MODulate:FREQuency?

- Description

  Set the modulation frequency for the specified channel's signal.

  <freq> represents frequency, the unit is Hz.

  <n>：Channel number, n takes value 1, 2.

- Return Format

  The query returns the modulation frequency of the specified channel's signal in scientific notation.

- For Example

  :CHANnel1:MODulate:FREQuency 2000

  Set the modulation frequency of CH1 to 2kHz.

  :CHANnel1:MODulate:FREQuency? The query returns 2e+3.

:CHANnel<n>:MODulate:ARB

- Syntax

  :CHANnel<n>:MODulate:ARB <source>,<filename>

  :CHANnel<n>:MODulate:ARB?

- Description

  Set the specified channel to load modulation arbitrary waveform data of a file under arbitrary wave source.

  <n>：Channel number, n takes value 1, 2.

  <source>：{INTernal|EXTernal }，which is internal and external.

  <filename>：The filename of arbitrary waveform.

- For Example

  :CHANnel1:MODulate:ARB INTernal,"AbsSine.bsv"

:CHANnel<n>:MODulate:DEPTh

- Syntax

  :CHANnel<n>:MODulate:DEPTh { <depth>}

  :CHANnel<n>:MODulate:DEPTh?

- Description

  Set the modulation depth for the specified channel.

  <depth> represents the modulation, the unit is %. The rang is 0% ~ 100%, the modulation depth of AM is 0%

~ 120%.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the modulation depth of the specified channel in scientific notation.

- For Example

:CHANnel1:MODulate:DEPTh 50 Set the modulation depth for CH1 to 50%.

:CHANnel1:MODulate:DEPTh? The query returns 5.000000e+01.

:CHANnel<n>:MODulate:RATio

- Syntax

:CHANnel<n>:MODulate:RATio <ratio>

:CHANnel<n>:MODulate:RATio?

- Description

Set the modulation rate value for the specified channel. This instruction is only valid for the modulation with rate function.

<ratio> represents rate, the unit is Hz.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the modulation rate value of the specified channel in scientific notation.

- For Example

:CHANnel1:MODulate:RATio 100 Set the bit rate for CH1 to 100Hz.

:CHANnel1:MODulate:RATio? The query returns 1.e+2.

:CHANnel<n>:FM:FREQuency:DEV

- Syntax

:CHANnel<n>:FM:FREQuency:DEV {<freq>}

:CHANnel<n>:FM:FREQuency:DEV?

- Description

Set the frequency deviation for the specified channel.

<freq> represents frequency deviation, the unit is Hz. The range is from 0Hz to the current fundamental frequency.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns the frequency deviation of the specified channel in scientific notation.

- For Example

:CHANnel1:FM:FREQuency:DEV 2000

Set the frequency deviation for CH1 to 2KHz.

:CHANnel1:FM:FREQuency:DEV? The query returns 2e+3.

:CHANnel<n>:PM:PHASe:DEV

- Syntax

  :CHANnel<n>:PM:PHASe:DEV {<phase>}

  :CHANnel<n>:PM:PHASe:DEV?

- Description

  Set the phase deviation for the specified channel output.

  <phase> represents phase deviation, the unit is°, the range is 0~360.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the phase deviation for the specified channel output.

- For Example

  :CHANnel1:PM:PHASe:DEV 30 Set phase deviation for CH1 to 30°.

  :CHANnel1:PM:PHASe:DEV? The query returns 30.

:CHANnel<n>:PWM:DUTY:DEV

- Syntax

  :CHANnel<n>:PWM:DUTY:DEV {<duty>}

  :CHANnel<n>:PWM:DUTY:DEV?

- Description

  Set the duty cycle deviation under PWM mode for the specified channel.

  < duty > represents duty cycle deviation, the unit is %. The range is 0-100.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the duty cycle deviation under PWM mode of the specified channel in scientific notation.

- For Example

  :CHANnel1:PWM:DUTY:DEV 10 Set duty cycle deviation for CH1 to 10%.

  :CHANnel1:PWM:DUTY:DEV? The query returns 1.e+1.

:CHANnel<n>:FSK:FREQuency

- Syntax

  :CHANnel<n>:FSK:FREQuency {<freq>}

  :CHANnel<n>:FSK:FREQuency?

- Description

  Set the hopping frequency of FSK for the specified channel. The modulation mode must be specified in advance for this command to take effect.

  < freq > represents frequency, the unit is Hz.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the hopping frequency of FSK for the specified channel in scientific notation.

- For Example

:CHANnel1:FSK:FREQ 2000 Set the hopping frequency of CH1 to 2KHz.
:CHANnel1:FSK:FREQ? The query returns 2e+3.

:CHANnel<n>:PSK:PHASe

- Syntax
  :CHANnel<n>:PSK:PHASe { < phase >}
  :CHANnel<n>:PSK:PHASe?
- Description
  Set the phase value of PSK for the specified channel. The modulation mode must be specified in advance for this command to take effect.
  < phase> represents phase, the unit is°. The range is 0~360.
  <n>: Channel number, n takes value 1, 2.
- Return Format
  The query returns the phase value of PSK for the specified channel in scientific notation.
- For Example
  :CHANnel1:PSK:PHAS 90 Set the output phase of CH1 to 90°.
  :CHANnel1:PSK:PHAS? The query returns 9e+1.

Frequency Sweep

:CHANnel<n>:SWEep:TYPe

- Syntax
  :CHANnel<n>:SWEep:TYPe { LINe|LOG }
  :CHANnel<n>:SWEep:TYPe?
- Description
  Set the frequency sweep mode for the specified channel. There are linear sweep and logarithmic sweep.
  <n>: Channel number, n takes value 1, 2.
- Return Format
  The query returns the frequency sweep mode of the specified channel.
- For Example
  :CHANnel1:SWEep:TYPe LINe Set the frequency sweep mode for CH1 to LINe.
  :CHANnel1:SWEep:TYPe? The query returns LINe.

:CHANnel<n>:SWEep:FREQuency:STARt

- Syntax
  :CHANnel<n>:SWEep:FREQuency:STARt <freq>
  :CHANnel<n>:SWEep:FREQuency:STARt?
- Description
  Set the start frequency for the frequency sweep of the specified channel.

< freq > represents frequency, the unit is Hz.

<n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the start frequency for the frequency sweep of the specified channel in scientific notation.

- For Example

  :CHANnel1:SWE:FREQ:STAR 2000

  Set the start frequency for the frequency sweep of CH1 to 2KHz.

  :CHANnel1:SWE:FREQ:STAR? The query returns 2e+3.

:CHANnel<n>:SWEep:FREQuency:STOP

- Syntax

  :CHANnel<n>:SWEep:FREQuency:STOP <freq>

  :CHANnel<n>:SWEep:FREQuency:STOP?

- Description

  Set the cut-off frequency for the frequency sweep of the specified channel.

  < freq > represents frequency, the unit is Hz.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the cut-off frequency for the frequency sweep of the specified channel in scientific notation.

- For Example

  :CHANnel1:SWE:FREQ:STOP 2000

  Set the cut-off frequency for the frequency sweep of CH1 to 2KHz.

  :CHANnel1:SWE:FREQ:STOP? The query returns 2e+3.

:CHANnel<n>:SWEep:TIMe

- Syntax

  :CHANnel<n>:SWEEP:TIMe <time>

  :CHANnel<n>:SWEEP:TIMe?

- Description

  Set the sweep time for the frequency sweep of the specified channel.

  < time > represents time, the unit is S. The range is 1ms ~ 500s.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the sweep time for the frequency sweep of the specified channel in scientific notation.

- For Example

  :CHANnel1:SWEEP:TIMe 2

  Set the sweep time for the frequency sweep of CH1 to 2S.

  :CHANnel1:SWEEP:TIMe? The query returns 2e+0.

Burst

:CHANnel<n>:BURSt:TYPe

- Syntax
  :CHANnel<n>:BURSt:TYPe {NCYC|GATe|INFinit}
  :CHANnel<n>:BURSt:TYPe?
- Description
  Set the burst mode for the specified channel. There are N cycle, gating, infinite.
  <n>: Channel number, n takes value 1, 2.
- Return Format
  The query returns the burst mode of the specified channel.
- For Example

:CHANnel1:BURSt:TYPe NCYC	Set the burst mode for CH1 to NCYC.
:CHANnel1:BURSt:TYPe?	The query returns 2e+0.

:CHANnel<n>:BURSt:PERiod

- Syntax
  :CHANnel<n>:BURSt:PERiod <period >
  :CHANnel<n>:BURSt:PERiod?
- Description
  Set the burst period for the specified channel.
  < period > represents time, the unit is S.
  <n>: Channel number, n takes value 1, 2.
- Return Format
  The query returns the burst period of the specified channel in scientific notation.
- For Example

:CHANnel1:BURSt:PERiod 5ms	Set the burst period for CH1 to 5ms.
:CHANnel1:BURSt:PERiod?	The query returns 5e-3.

:CHANnel<n>:BURSt:PHASe

- Syntax
  :CHANnel<n>:BURSt:PHASe <phase>
  :CHANnel<n>:BURSt:PHASe?
- Description
  Set phase of burst for the specified channel.
  < phase > represents phase, the unit is°. The range is 0 ~ 360.
  <n>: Channel number, n takes value 1, 2.
- Return Format
  The query returns phase of burst for the specified channel in scientific notation.

- For Example

  :CHANnel1:BURSt:PHASe 18 Set phase of burst for CH1 to 18°.

  :CHANnel1:BURSt:PHASe? The query returns 1.8e+1.

:CHANnel<n>:BURSt:CYCLes

- Syntax

  :CHANnel<n>:BURSt:CYCLes <cycles>

  :CHANnel<n>:BURSt:CYCLes?

- Description

  Set cycle of burst for the specified channel.

  < cycles >represents cycle numbers, it is integer data.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns cycle of burst for the specified channel in scientific notation.

- For Example

  :CHANnel1:BURSt:CYCLes 2

  Set cycle of burst for the specified channel to 2.

  :CHANnel1:BURSt:CYCLes? The query returns 2

:CHANnel<n>:BURSt:GATe:POLarity

- Syntax

  :CHANnel<n>:BURSt:GATe:POLarity {POSitive|NEGative}

  :CHANnel<n>:BURSt:GATe:POLarity?

- Description

  Set the polarity of gating for the specified channel. There are positive and negative.

  <n>: Channel number, n takes value 1, 2.

- Return Format

  The query returns the polarity of gating for the specified channel.

- For Example

  :CHANnel1:BURSt:GATe:POLarity POSitive Set the polarity of gating for CH1 to POSitive.

  :CHANnel1:BURSt:GATe:POLarity? The query returns POSitive.

:CHANnel<n>:TRIGger:SOURce

- Syntax

  :CHANnel<n>:TRIGger:SOURce {INTernal|EXTErnal}

  :CHANnel<n>:TRIGger:SOURce?

- Description

  Set trigger source for the specified channel. This instruction is only valid for burst function.

  <n>: Channel number, n takes value 1, 2.

- Return Format

The query returns trigger source of the specified channel.

- For Example

:CHANnel1:TRIGger:SOURce INTernal Set trigger source of CH1 to INTernal.

:CHANnel1:TRIGger:SOURce? The query returns INTernal.

:CHANnel<n>:BURSt:TRGEdge

- Syntax

:CHANnel<n>:BURSt:TRGEdge {RISe|FALL}

:CHANnel<n>:BURSt: TRGEdge?

- Description

Set the trigger edge of burst mode for the specified channel. This instruction is only valid for external trigger of burst function.

<n>: Channel number, n takes value 1, 2.

- Return Format

The query returns trigger output mode for the specified channel.

- For Example

:CHANnel1:BURSt:TRGEdge RISe

Set trigger output of the rising edge for CH1 to RISe.

:CHANnel1:BURSt:TRGEdge? The query returns RISe.

WARB Command

It is used to write arbitrary wave file command, including write configuration of basic arbitrary wave and modulation arbitrary wave.

:WARB<n>:MODulate

- Syntax

:WARB<n>:MODulate <arb file>

- Description

It is used to write modulation arbitrary wave, wave data fix at 4k points. Send this instruction at first and then send the arbitrary wave file data to signal source.

<arb file> represents arbitrary wave file.

- For Example

:WARB1:MODulate "GateVibar.bsv" Write modulation arbitrary wave file for CH1.

:WARB<n>:CARRier

- Syntax

:WARB<n>:CARRier <arb file>

- Description

It is used to write basic arbitrary wave, wave data fix at 4k points. Send this instruction at first and then send the arbitrary wave file data to signal source.

<arb file> represents arbitrary wave file.

➢ For Example

:WARB1: CARRier "GateVibar.bsv" Write basic arbitrary wave file for CH1.

DISPlay Command

It is used to acquire the information of signal source.

:DISPlay?

➢ Syntax

:DISPlay?

➢ Description

To query the currently image data on the screen of signal source.

➢ Return Format

The query returns the image data, return data is conform to Appendix 2: IEEE 488.2 Binary Data Format.

➢ For Example

:DISPlay? The query returns the image data.

Explanation of Programming

This chapter is to describe troubleshooting in process of programming. If you meet any of the following problems, please handle them according to the related instructions.

Programming Preparation

User can remote the spectrum analyzer via USB or LAN interface on the spectrum analyzer and combine with the NI-VISA and programming language. Programming preparation is only applicable for using Visual Studio and LabVIEW development tools to programming under Windows operating system.

1. Setup Communication

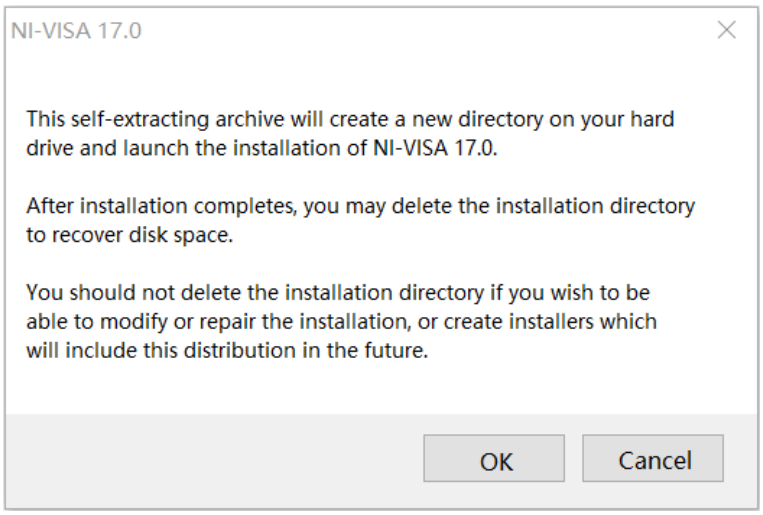
NI-VISA is the communication library for communication between computers and devices. NI software has two valid VISA installation packages: Full version and the Run-Time Engine version. The full version includes the NI device driver and the NI MAX tool, NI MAX is the used to control user interface of the device. While the driver and NI MAX are useful, they are not used for remote control. Run-Time Engine is a smaller file than the full version, and it is primarily used for remote control.

You can download the latest NI-VISA Run-Time Engine or the full version from the NI website. The installation steps are basically the same.

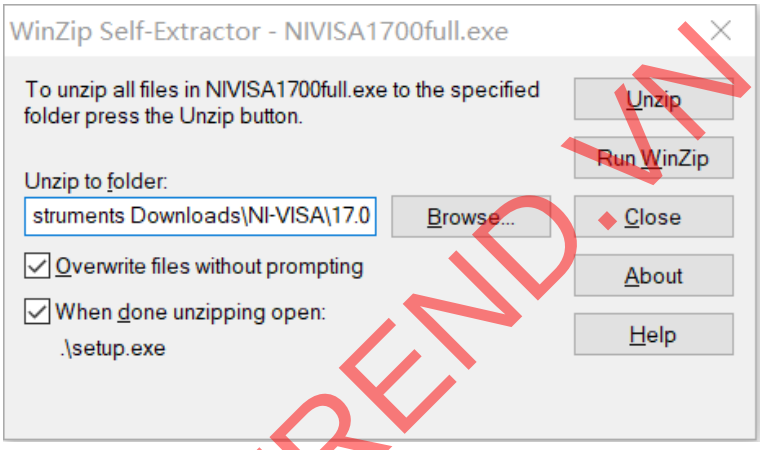
Follow these steps to install NI-VISA (take NI-VISA 17.0 full version as an example).

a. Download the proper version of NI-VISA;

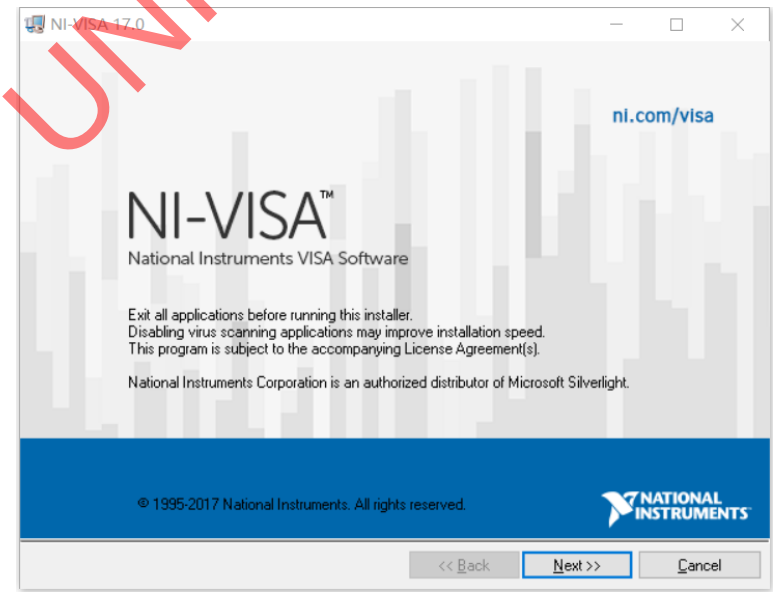
b. Double click NIVISA1700full.exe to pop-out the dialogue,



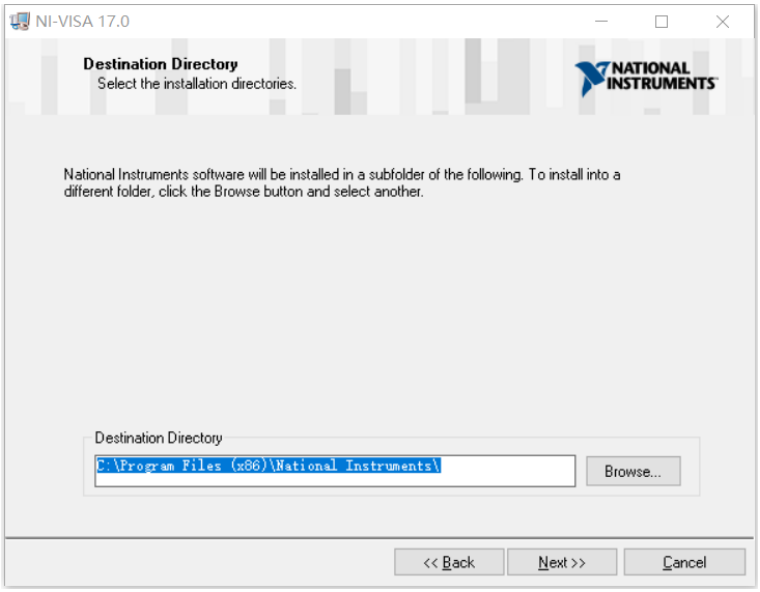
c. Click to confirm to pop-out the dialogue,



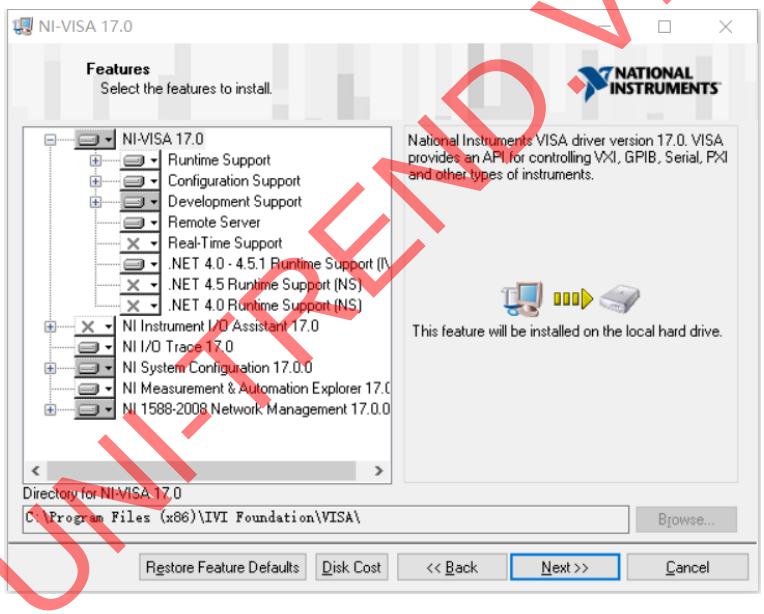
d. Click Unzip to decompressing files, after the decompression is completed, the installation program will execute automatically. If your computer needs to install .NET Framework4, it will installed automatically during the installation.



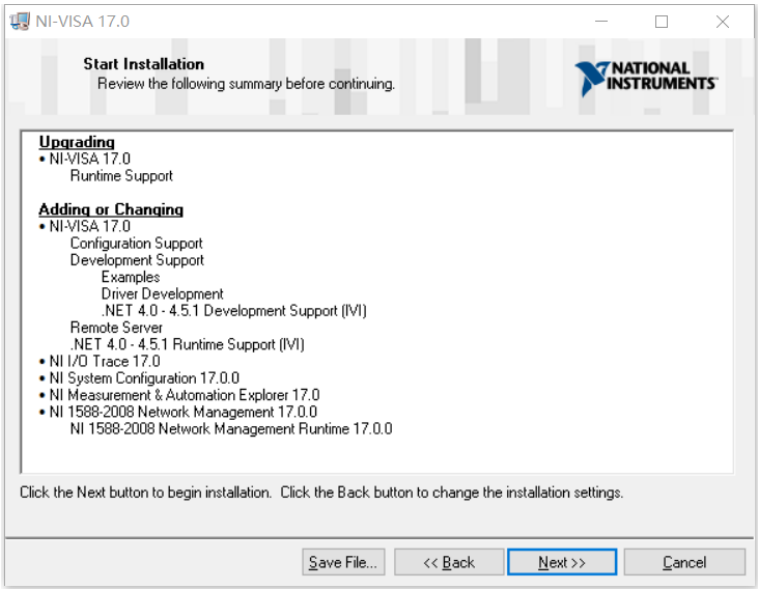
e. Installation dialogue of NI-VISA as shown in the figure above. Click Next to start the installation.



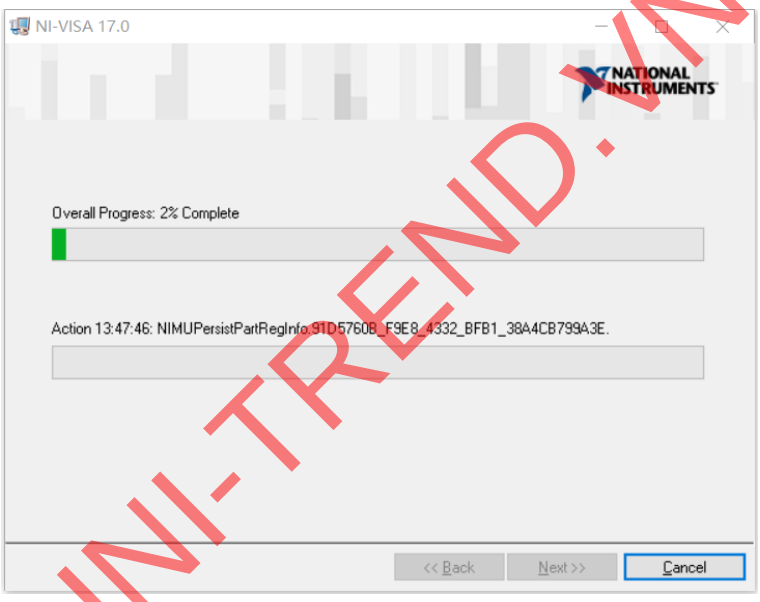
f. To set the installation path, the default path is "C:\Program Files (x86)\National Instruments\", or you can click Next to change the installation path, as shown in the following figure,



g. Double click Next in the license agreement dialogue and to select"I accept the above 2 License Agreement(s)." and then click Next, the dialogue as shown in the following figure,



h. Click Next to start the installation.

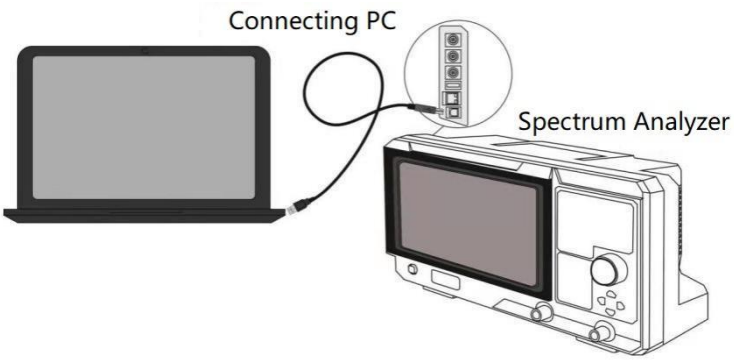


j. Restart the computer after the installation is completed.

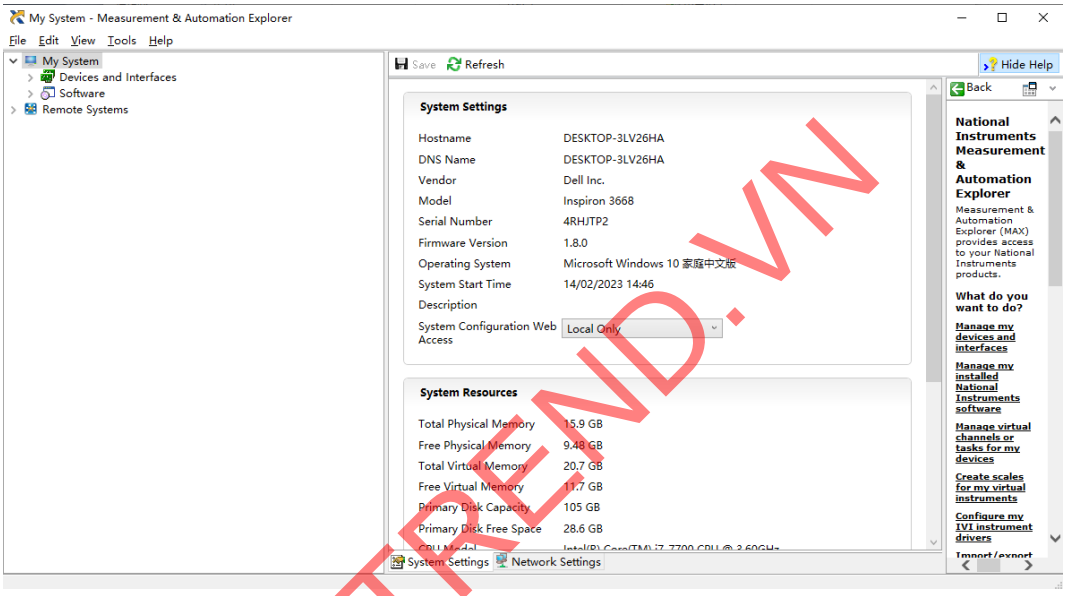
2. Connecting Device

Take USB method as an example to introduce the connection.

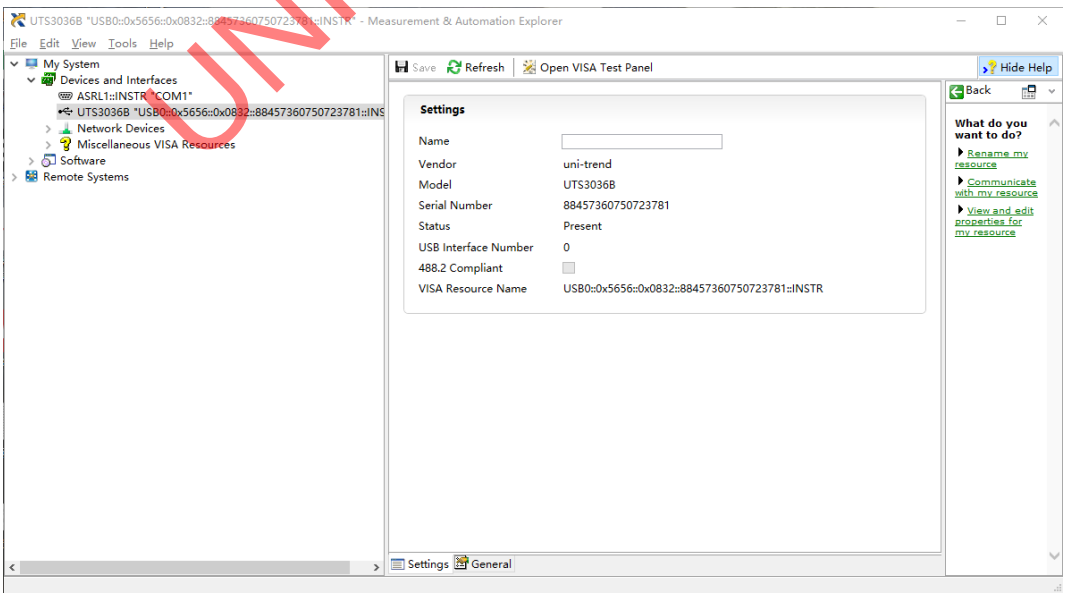
a. Turn on the spectrum analyzer;
b. Use USB wire to connect the USB Device port of the spectrum analyzer with USB Host port of the computer, as shown in the following figure,



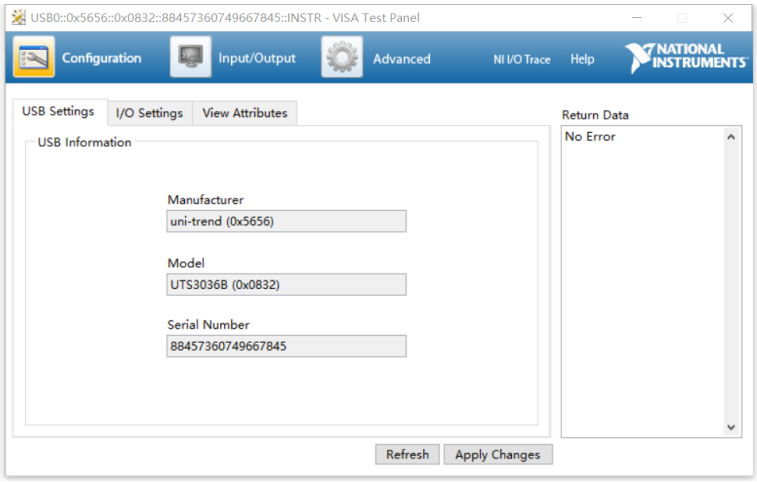
c. Turn on NI MAX on the computer. The dialogue as shown in the following figure,



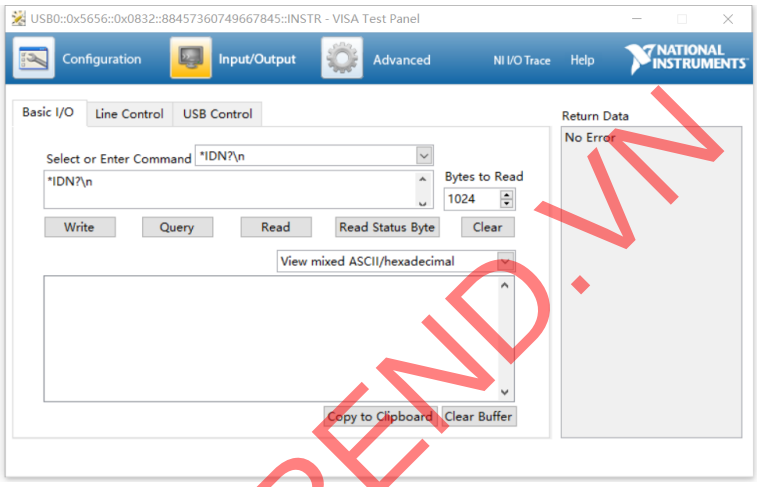
d. Turn on the device and scroll down the options to select the spectrum analyzer's drive, as shown in the following figure,



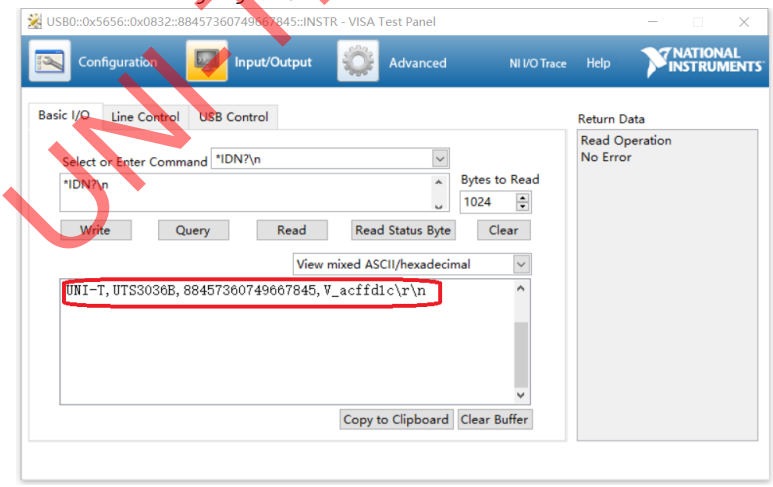
e. Use mouse to click VISA test panel to pop-out the dialogue as shown in the following figure,



f. Use mouse to click the option Input/Output, as shown in the following figure,



g. Use mouse to click Query to query IDN of the spectrum analyzer, the query result will display at the red area, as shown in the following figure,



h. If it can query the relevant information of the spectrum analyzer, which means of the spectrum analyzer is communication with the computer.

VISA Programming Example

There are some examples in this section. Through these examples, user can know how to use VISA, and it can combine with the command of programming manual to realize the control of the instrument. With these examples, user can develop more applications.

VC++ Example

- Environment: Window system, Visual Studio
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps

1. Open Visual Studio software to create a new VC++ win32 console project.
2. Set project environment that can adjust NI-VISA libray, which are static library and dynamic library.

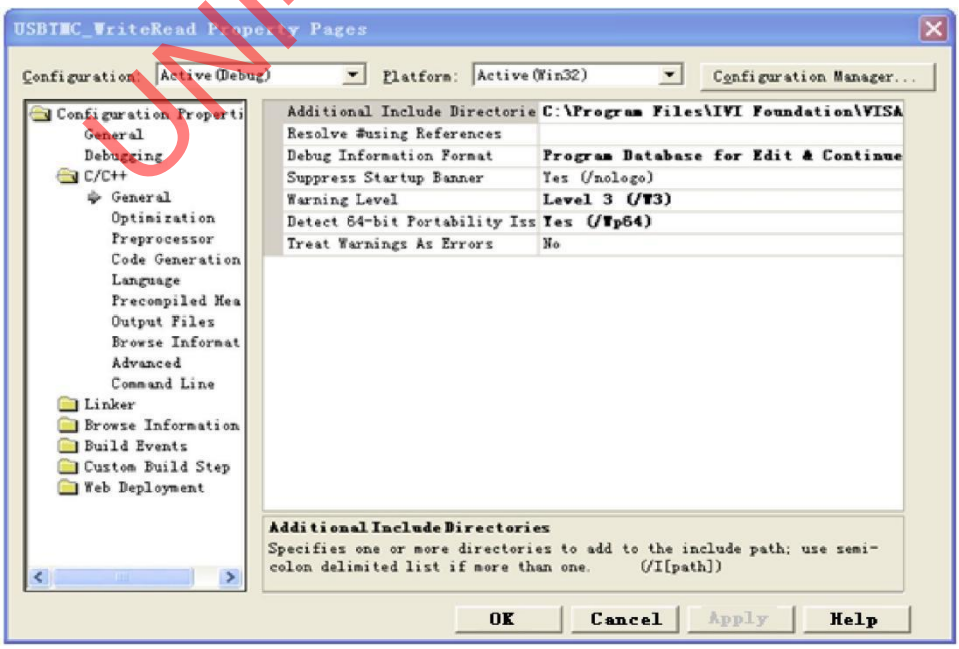
a) Static library:

In NI-VISA installment path to find file visa.h, visatype.h and visa32.lib and copy them to the root path of VC++ project and add it to the project. Add two lines of code into file projectname.cpp as follows.

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

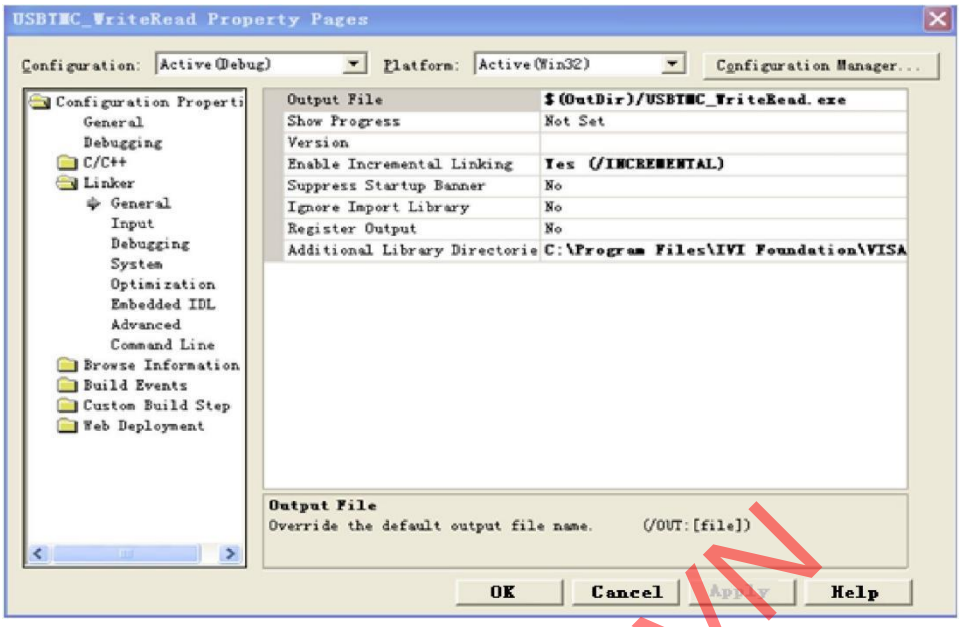
b) Dynamic library:

Press "project>>properties", select "c/c++---General" in attribute dialog on the left side, set the value of "Additional Include Directories" as the installment path of NI-VIS (such as C:\ProgramFiles\IVI Foundation\VISA\WinNT\include), as shown in the following figure.

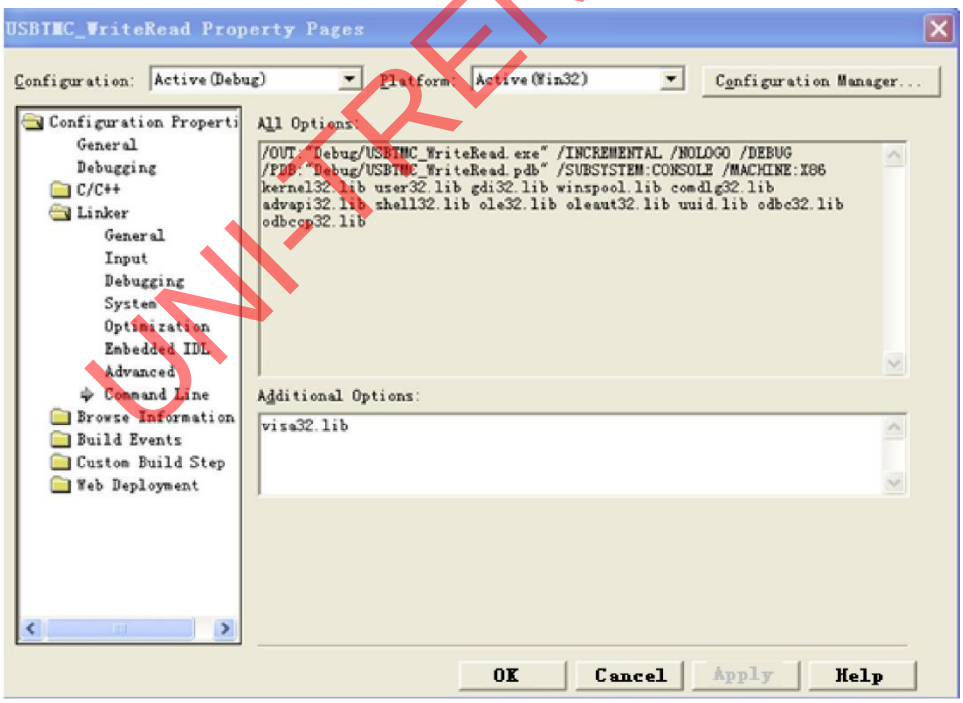


Select "Linker-General" in attribute dialog on the left side, set the value of "Additional Library Directories" as the installment path of NI-VIS (such as C:\Program Files\IVI

Foundation\VISA\WinNT\include), as shown in the following figure.



Select "Linker-General" in attribute dialog on the left side, set the value of "Additional Library Directories" to visa32.lib, as shown in the following figure.



Add file visa.h in projectname.cpp file,

#include <visa.h>

1. Source code

a) USBTMC Example

```
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.
    * 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;
    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;
}
```

b) TCP/IP Example

```
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[ ] = "::inst0::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;
```

C# Example

- Environment: Window system, Visual Studio
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open Visual Studio software and create a new C# console project.
2. Add C# quote Ivi.Visa.dll and NationalInstruments.Visa.dll of VISA.
3. Source code

a) USBTMC Example

```
class Program
{
    void usbtmc_test()
    {
        using (var rmSession = new ResourceManager())
        {
            var resources = rmSession.Find("USB?*INSTR");
            foreach (string s in resources)
            {
                try
                {
                    var mbSession = (MessageBasedSession)rmSession.Open(s);
                    mbSession.RawIO.Write("*IDN?\n");
                    System.Console.WriteLine(mbSession.RawIO.ReadString());
                }
```

```
                catch (Exception ex)
                {
                    System.Console.WriteLine(ex.Message);
                }
            }
        }
    }

    void Main(string[ ] args)
    {
        usbtmc_test();
    }
}
```

b) TCP/IP Example

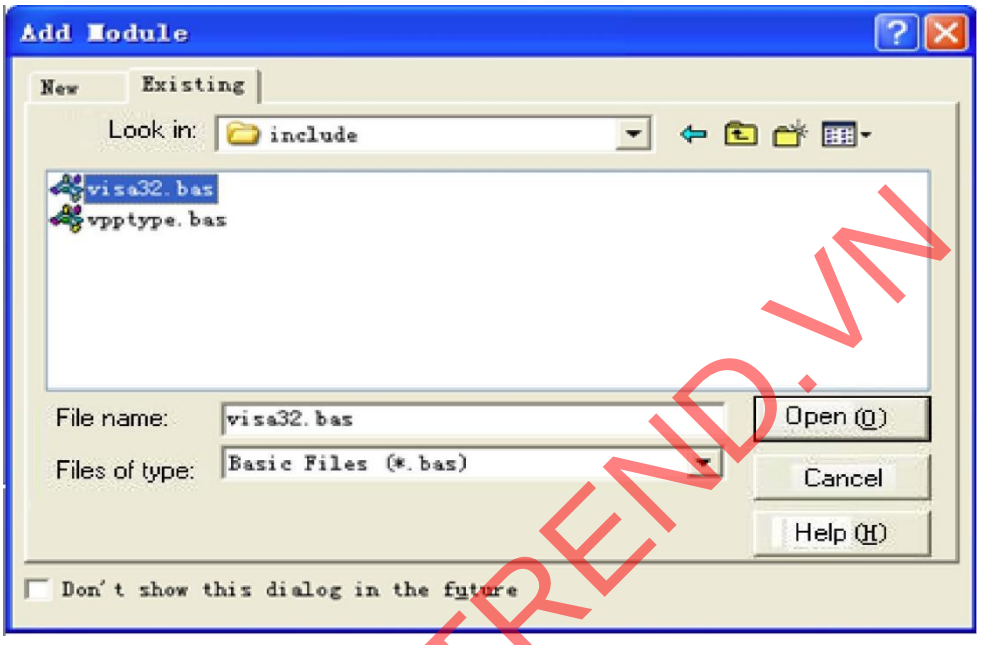
```
class Program
 {
    void tcp_ip_test(string ip)
    {
        using (var rmSession = new ResourceManager())
        {
            try
            {
                var resource = string.Format("TCPIP0::{0}::inst0::INSTR", ip);
                var mbSession = (MessageBasedSession)rmSession.Open(resource);
                mbSession.RawIO.Write("*IDN?\n");
                System.Console.WriteLine(mbSession.RawIO.ReadString());
            }
            catch (Exception ex)
            {
                System.Console.WriteLine(ex.Message);
            }
        }
    }

    void Main(string[ ] args)
    {
        tcp_ip_test("192.168.20.11");
    }
```

VB Example

- Environment: Window system, Microsoft Visual Basic 6.0.
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open Visual Basic software and create a new standard application program project.
2. Set the project environment that can adjust NI-VISA library, press Existing tab of Project>>Add Existing Item, in file "include" of NI-VISA installment path to find file visa32.bas and add this file, as shown in the following figure.



3. Source code

a) USBTMC Example

```
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 Example

```
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
```

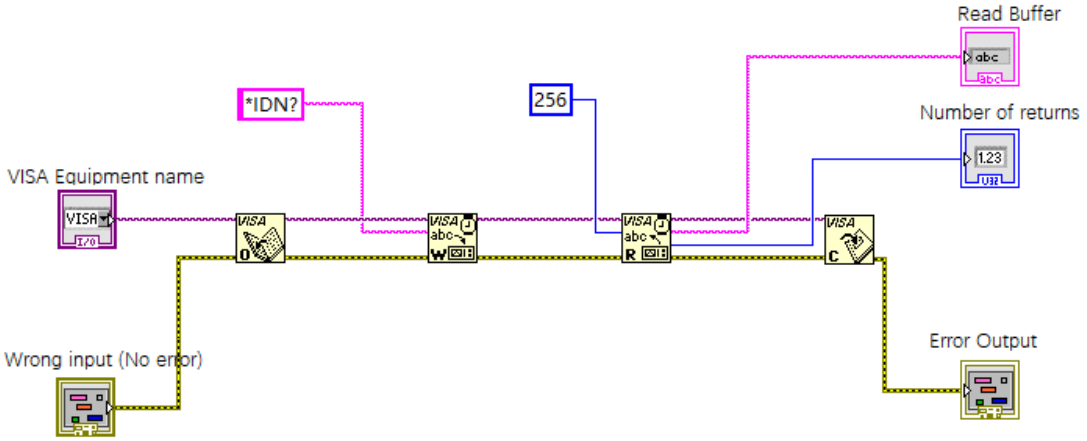
```
EndIf

' Now we will open a session via TCP/IP device
status = viOpen(defaultRM, "TCPIP0::" + ip + "::inst0::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
```

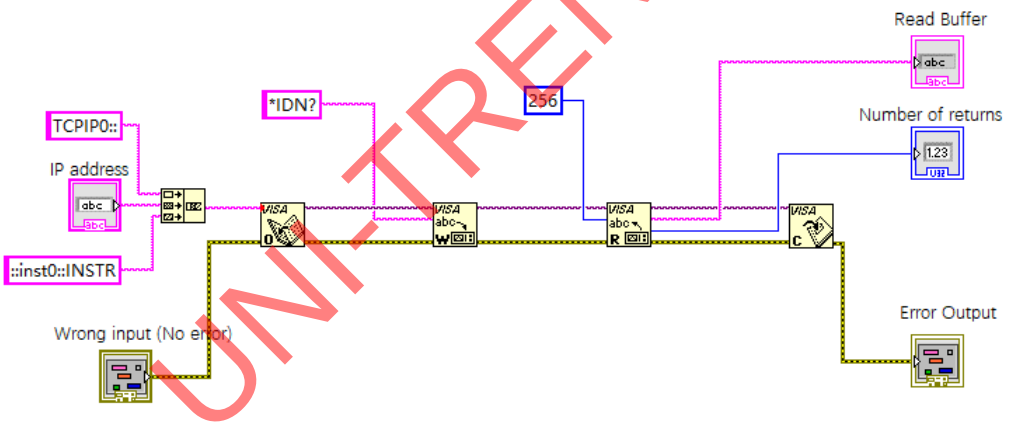
LabVIEW Example

- Environment: Window system, LabVIEW
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open LabVIEW software and create a VI file.
2. Add control, press the front panel interface, select and add VISA resource name, error input, error output and partial indetifier on control flow diagram.
3. Open diagram, press VISA resource name and then select and add function VISA Write, VISA Read, VISA Open and VISA Close on pop-out menu.
4. VI open a VISA session of USBTMC device and wrote *IDN? command and read back the response value. When all communication is complete, VI will close the VISA session. As shown in the following figure.



5. Communication with the device via TCP/IP is similar with USBTMC, it need to set VISA write and read function to synchronous I/O, set LabVIEW to asynchronous IO by default. Right click on the node and select "Synchronous I/O Mode>>Synchronous" from shortcut menu to enable synchronous writing or reading of data, as shown in the following figure.



MATLAB Example

- Environment: Window system, MATLAB
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open MATLAB software, click File>>New>>Script on Matlab interface to create an empty M file.
2. Source code

a) USBTMC Example

```
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::0x5345::0x1234::SN20220718::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
```

b) TCP/IP Example

```
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::','192.168.20.11','::inst0::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
```

Python Example

- Environment: Window system, Python3.8, PyVISA 1.11.0.
- Description: Access the instrument via USBTMC and TCP/IP, send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Install python, and then turn on Python script compiling software, create an empty test.py file.
2. Use pip install PyVISA instruction to install PyVISA, if it cannot install, please refer to this link (https://pyvisa.readthedocs.io/en/latest/).
3. Source code

a) USBTMC Example

```
import pyvisa
rm = pyvisa.ResourceManager()
rm.list_resources()
my_instrument = rm.open_resource('USB0::0x5345::0x1234::SN20220718::INSTR')
print(my_instrument.query('*IDN?'))
```

b) TCP/TP Example

```
import pyvisa
rm = pyvisa.ResourceManager()
rm.list_resources()
my_instrument = rm.open_resource('TCPIP0::192.168.20.11::inst0::INSTR')
print(my_instrument.query('*IDN?'))
```

Programming Application Example

Configuring Sine Wave

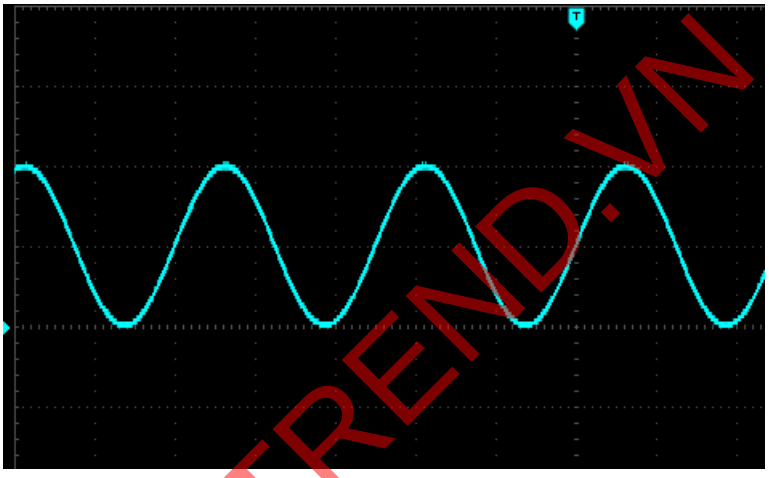
This section is to introduce how to configure the sine wave function.

Explanation

A sine wave has an amplitude, an offset, and a phase relative to a synchronous pulse. It can use high voltage value and low voltage value to set its amplitude and deviation.

Example

The following wave can set by SCPI command, high voltage value and low voltage value can replace :CHANnel1:BASE:AMPLitude and :CHANnel1:BASE:OFFSet



The following command can generate the sine wave as shown above.

```
:CHANnel1:MODe CONTinue
:CHANnel1:BASE:WAVe SINe
:CHANnel1:BASE:FREQuency 2000
:CHANnel1:BASE:HIGH 2
:CHANnel1:BASE:LOW 0
:CHANnel1:BASE:PHAse 20
:CHANnel1:OUTPut ON
```

Configuring Square Wave

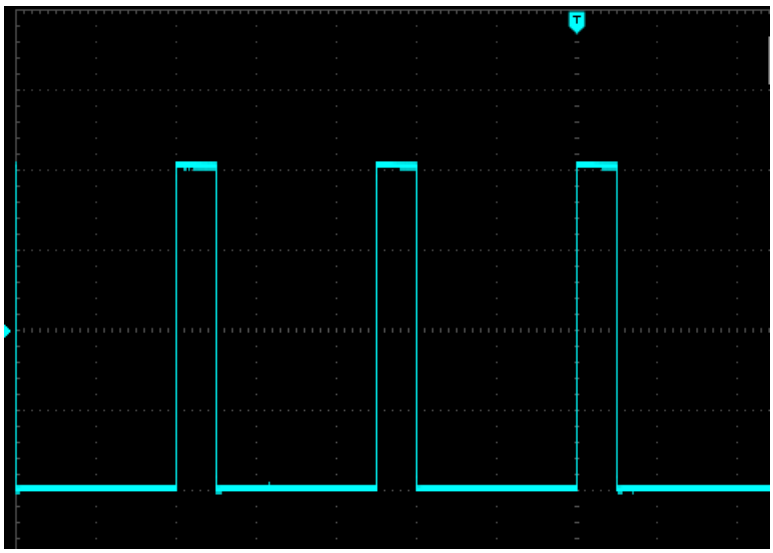
This section is to introduce how to configure the square wave.

Explanation

A square wave has an amplitude, an offset, and a phase relative to a synchronous pulse. It also has duty cycle and period. It can use high voltage value and low voltage value to set its amplitude and deviation.

Example

The following wave can set by SCPI command.



The following command can generate the square wave as shown above.

:CHANnel1:MODe CONTinue
:CHANnel1:BASE:WAVe SQUare
:CHANnel1:BASE:FREQuency 40000
:CHANnel1:BASE:AMPLitude 2
:CHANnel1:BASE:OFFSet 0
:CHANnel1:BASE:PHAse 90
:CHANnel1:BASE:DUTY 20
:CHANnel1:OUTPut ON

Configuring Sawtooth Wave

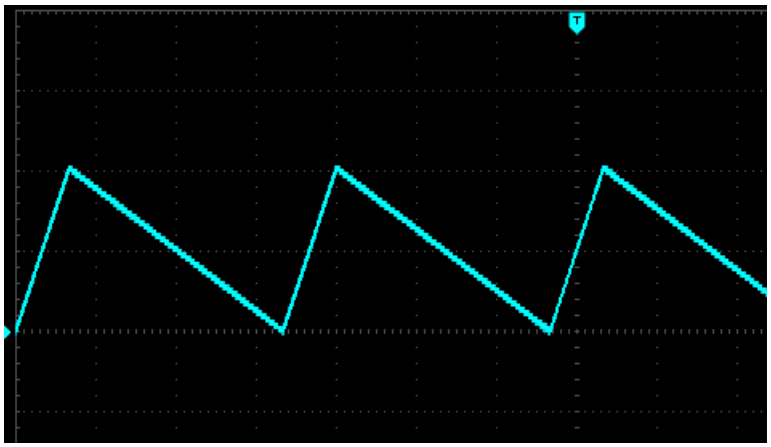
This section is to introduce how to configure the sawtooth wave.

Explanation

A sawtooth wave has an amplitude, an offset, and a phase relative to a synchronous pulse. It also can create triangle wave and the symmetry of other similar waves. It can use high voltage value and low voltage value to set its amplitude and deviation.

Example

The following wave can set by SCPI command, high voltage value and low voltage value can replace : CHANnel1:BASE:AMPLitude and :CHANnel1:BASE:OFFSet



The following command can generate the sawtooth wave as shown above.

:CHANnel1:MODe CONTinue
:CHANnel1:BASE:WAVe RAMP
:CHANnel1:BASE:FREQuency 30000
:CHANnel1:BASE:HIGH 2
:CHANnel1:BASE:LOW 0
:CHANnel1:BASE:PHAse 90
:CHANnel1:RAMP:SYMMetry 20
:CHANnel1:OUTPut ON

Configuring Impulse Wave

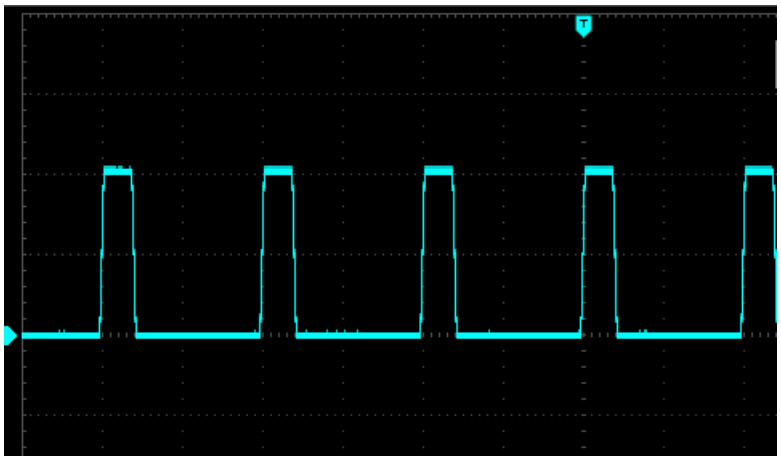
This section is to introduce how to configure the impulse wave.

Explanation

An impulse wave has an amplitude, an offset, and a phase relative to a synchronous pulse. It also adds slope of edge and duty cycle (or pulse width). It can use high voltage value and low voltage value to set its amplitude and deviation.

Example

The following wave can set by SCPI command, high voltage value and low voltage value can replace :CHANnel1:BASE:AMPLitude and :CHANnel1:BASE:OFFSet



The following command can generate the impulse wave as shown above.

```
:CHANnel1:MODe CONTinue
:CHANnel1:BASE:WAVe PULSe
:CHANnel1:BASE:FREQuency 100000
:CHANnel1:BASE:HIGH 2
:CHANnel1:BASE:LOW 0
:CHANnel1:BASE:PHAse 270
:CHANnel1:BASE:DUTY 20
:CHANnel1:PULSe:RISe 0.0000002
:CHANnel1:PULSe:FALL 0.0000002
:CHANnel1:OUTPut ON
```

Configuring Arbitrary Wave

This section is to introduce how to configure the arbitrary wave.

Explanation

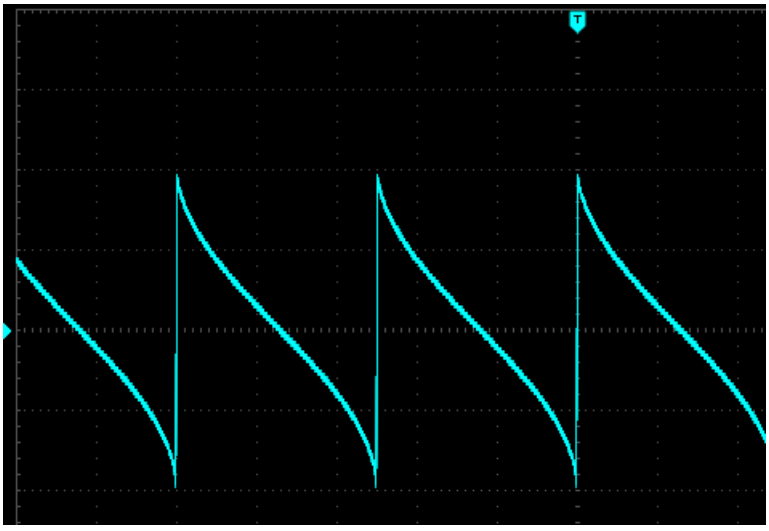
A harmonic wave has frequency, amplitude, offset and phase. It also adds mode and wave file.

Example

The following code can load and modify built-in arbitrary wave.

```
:CHANnel1:MODe CONTinue
:CHANnel1:BASE:WAVe ARB
:CHANnel1:ARB:MODe DDS
:CHANnel1:BASE:ARB INTernal,"ACos.bsv"
:CHANnel1:BASE:FREQuency 200000
:CHANnel1:BASE:AMPLitude 2
:CHANnel1:BASE:OFFSet 0
:CHANnel1:BASE:PHAse 90
:CHANnel1:OUTPut ON
```

The wave generate by the above command as shown below.



Appendix 1：Key List

Key	Function Description	LED
Wave	Wave mode	
Mode	Output mode	√
Utility	System	
Symbol	Digit key symbol	
Dot	Digit key decimal point	
NUM0	Digit key 0	
NUM1	Digit key 1	
NUM2	Digit key 2	
NUM3	Digit key 3	
NUM4	Digit key 4	
NUM5	Digit key 5	
NUM6	Digit key 6	
NUM7	Digit key 7	
NUM8	Digit key 8	
NUM9	Digit key 9	
Up	Arrow key UP	
Down	Arrow key DOWN	
Left	Arrow key LEFT	
Right	Arrow key RIGHT	
OK	Confirm key	
CH1	Channel 1 key	√
CH2	Channel 2 key	√
F1	Select the first menu option at the current menu	
F2	Select the second menu option at the current menu	
F3	Select the third menu option at the current menu	
F4	Select the fourth menu option at the current menu	
F5	Select the fifth menu option at the current menu	
F6	Select the sixth menu option at the current menu	

Appendix 2: IEEE 488.2 Binary Data Format

DATA is data flow, other is ASCII, as shown in the following table

<#812345678 + DATA + \n>

Start Mark（1Byte）	Length of Bit Width（1Byte）	Total Length of Data（Bit Width Byte）	DATA（n Byte）	End Mark（1Byte）
#	x	xxxxxxxx	.....	\n