

# UTG1000X系列函数/任意波形发生器

## 编程手册

REV 0  
2022.09

**UNI-T**<sup>®</sup>

# 保证和声明

## 版权

2017 优利德中国科技有限公司

## 商标信息

UNI-T是优利德中国科技有限公司的注册商标。

## 文档编号

## 软件版本

3. 07. 000

软件升级可能更改或增加产品功能，请关注 **UNI-T**网站获取最新版本手册或联系 **UNI-T**升级软件。

## 声明

- 本公司产品受中国及其它国家和地区的专利（包括已取得的和正在申请的专利）保护。
- 本公司保留改变规格及价格的权利。
- 本手册提供的信息取代以往出版的所有资料。
- 本手册提供的信息如有变更，恕不另行通知。
- 对于本手册可能包含的错误，或因手册所提供的信息及演绎的功能以及因使用本手册而导致的任何偶然或继发的损失，**UNI-T**概不负责。
- 未经 **UNI-T**事先书面许可，不得影印、复制或改编本手册的任何部分。

## 产品认证

**UNI-T**认证本产品符合中国国家产品标准和行业产品标准及 ISO9001：2008 标准和 ISO14001：2004 标准，并进一步认证本产品符合其它国际标准组织成员的相关标准。

## 联系我们

如您在使用此产品或本手册的过程中有任何问题或需求，可与 **UNI-T**联系：

电子邮箱：

网址：[www.uni-trend.com](http://www.uni-trend.com)

# SCPI 指令简介

SCPI (Standard Commands for Programmable Instruments, 即可编程仪器标准命令集) 是一种建立在现有标准 IEEE 488.1 和 IEEE 488.2 基础上, 并遵循了 IEEE754 标准中浮点运算规则、ISO646 信息交换 7 位编码符号 (相当于 ASCII 编程) 等多种标准的标准化仪器编程语言。本节简介 SCPI 命令的格式、符号、参数和缩写规则。

## 指令格式

SCPI 命令为树状层次结构, 包括多个子系统, 每个子系统由一个根关键字和一个或数个层次关键字构成。命令行通常以冒号 “:” 开始; 关键字之间用冒号 “:” 分隔, 关键字后面跟随可选的参数设置。命令关键字和第一个参数之间以空格分开。命令字符串必须以一个 <换行> (<NL>) 字符结尾。命令行后面添加问号 “?” 通常表示对此功能进行查询。

## 符号说明

下面四种符号不是 SCPI 命令中的内容, 不随命令发送, 但是通常用于辅助说明命令中的参数。

- **大括号 { }**  
大括号中通常包含多个可选参数, 发送命令时必须选择其中一个参数。如: `DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE}` 命令。
- **竖线 |**  
竖线用于分隔多个参数选项, 发送命令时必须选择其中一个参数。  
如: `DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE}` 命令。
- **方括号 [ ]**  
方括号中的内容 (命令关键字) 是可省略的。如果省略参数, 仪器将该参数设置为默认值。例如: 对于 `MEASure:NDUTy? [<source>]` 命令, `[<source>]` 表示当前通道。
- **三角括号 < >**  
三角括号中的参数必须用一个有效值来替换。例如: 以 `DISPlay:GRID:BRIGhtness 30` 的形式发送 `DISPlay:GRID:BRIGhtness <count>` 命令。

## 参数说明

本手册介绍的命令中所含的参数可以分为以下 5 种类型: 布尔型、整型、实型、离散型、ASCII 字符串。

- **布尔型**  
参数取值为 “ON” (1) 或 “OFF” (0)。例如: `:SYSTem:LOCK {{1 | ON} | {0 | OFF}}`。
- **整型**  
除非另有说明, 参数在有效值范围内可以取任意整数值。注意: 此时, 请不要设置参数为小数格式, 否则将出现异常。例如: `:DISPlay:GRID:BRIGhtness <count>` 命令中的参数 `<count>` 可取 0 到 100 范围内的任一整数。

- **实型**

除非另有说明，参数在有效值范围内可以取任意值。

例如：对于 CH1, CHANnel1:OFFSet <offset>命令中的参数<offset>的取值为实型。

- **离散型**

参数只能取指定的几个数值或字符。例如：:DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE} 命令的参数只能为 FULL、GRID、CROSS、NONE。

- **ASCII 字符串**

字符串参数实际上可包含所有 ASCII 字符集。字符串必须以配对的引号开始和结尾；可以用单引号或双引号。引号分隔符也可以作为字符串的一部分，只需键入两次并且不在中间添加任何字符，例如设置 IP: SYST:COMM:LAN:IPAD "192.168.1.10"。

## 简写规则

所有命令对大小写都能识别，可以全部采用大写或小写。如果要缩写，必须输完命令格式中的所有大写字母。

## 数据返回

数据返回分为单个数据和批量数据返回，单个数据返回相对应的参数类型，其中实型数据返回用科学计数法表示，e 前部分小数点后面保留三位数据，e 部分保留三位数据；批量数据返回必须符合 IEEE 488.2 #格式的字符串数据，其格式：‘#’ + 长度所占的字符位数[固定为一个字符] + 有效数据长度的 ASCII 值 + 有效数据 + 结束符[‘\n’]，例如#3123xxxxxxxxxxxxxxxxxxxx\n 表示的具有 123 个字节有效批量数据返回格式，其中‘3’表示“123”占 3 个字符位。

# SCPI 指令详解

## IEEE488.2 通用命令

### \*IDN?

➤ **命令格式:**

\*IDN?

➤ **功能描述:**

用于查询制造商名称、信号源型号、产品序列号和软件版本号。

➤ **返回格式:**

制造商名称, 信号源型号、产品序列号, 由点号分隔的软件版本号。

➤ **举例:**

UNI-T Technologies, UTG1000X, 000000001, 00.00.01

### \*RST

➤ **命令格式:**

\*RST

➤ **功能描述:**

用于恢复出厂设置并清空所有的错误信息及发送接收队列缓冲。

## SYSTem 命令

用于对信号源进行最基本的操作, 主要包括语音选择、系统设置数据等操作。

### :SYSTem:CONFIgure

➤ **命令格式:**

:SYSTem:CONFIgure <file>

:SYSTem:CONFIgure?

➤ **功能描述:**

用于读写配置文件, 先发送该指令, 然后发送配置文件数据到信号源。

<file>表示配置文件。

➤ **返回格式:**

查询返回信号源当前配置文件数据。

➤ **举例:**

:SYSTem:CONFIgure

写入配置文件数据到信号源中并使其加载

:SYSTem:CONFIgure?

查询返回信号源当前配置文件数据二进制流

### :SYSTem:PHASe:MODE

➤ **命令格式:**

:SYSTem:PHASe:MODE {INDePendent | SYNChronization}

:SYSTem:PHASe:MODE?

➤ **功能描述:**

控制通道间的相位模式，若为同步，则表示两个通道起始相位保持同步，否则相位独立。

➤ **返回格式:**

查询返回通道间的相位模式。

➤ **举例:**

:SYSTem:PHASe:MODE INDePendent                    设置通道间为独立相位模式

:SYSTem:PHASe:MODE?                                查询返回 INDePendent

### :SYSTem:LANGUage

➤ **命令格式:**

:SYSTem:LANGUage {ENGLi sh|CHINese}

:SYSTem:LANGUage?

➤ **功能描述:**

控制系统显示语言。

➤ **返回格式:**

查询返回系统显示语言。

➤ **举例:**

:SYSTem:LANGUage ENGLi sh                        设置英文为系统显示语言

:SYSTem:LANGUage?                                查询返回 ENGLi sh

### :SYSTem:BEEP

➤ **命令格式:**

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

:SYSTem:BEEP?

➤ **功能描述:**

控制系统蜂鸣器开关

➤ **返回格式:**

查询返回蜂鸣器开关状态。

➤ **举例:**

:SYSTem:BEEP ON                                打开蜂鸣器

:SYSTem:BEEP?                                查询返回 1

## :SYSTem:NUMBer:FORMat

### ➤ 命令格式:

:SYSTem:NUMBer:FORMat {COMMA|SPACE|NONE}

:SYSTem:NUMBer:FORMat?

### ➤ 功能描述:

控制系统数字格式的分隔符

### ➤ 返回格式:

查询返回系统数字格式的分隔符。

### ➤ 举例:

:SYSTem:NUMBer:FORMat NONE                    设置无系统数字格式

:SYSTem:NUMBer:FORMat?                        查询返回 NONE

## :SYSTem:BRIGhtness

### ➤ 命令格式:

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

:SYSTem:BRIGhtness?

### ➤ 功能描述:

控制系统背光亮度等级

### ➤ 返回格式:

查询返回系统背光亮度等级

### ➤ 举例:

:SYSTem:BRIGhtness 30                        设置系统背光亮度 30%

:SYSTem:BRIGhtness?                        查询返回 30

## :SYSTem:SLEEP:TIME

### ➤ 命令格式:

:SYSTem:SLEEP:TIME {CLOSE|1MIN|5MIN|15MIN|30MIN|60MIN}

:SYSTem:SLEEP:TIME?

### ➤ 功能描述:

控制系统休眠时间，单位是分钟

### ➤ 返回格式:

查询返回休眠时间

### ➤ 举例:

:SYSTem:SLEEP:TIME 5 MIN                    设置系统 5 分钟之后自动休眠

:SYSTem:SLEEP:TIME?                        查询返回 5MIN

## :SYSTem:CYMometer

### ➤ 命令格式:

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

:SYSTem:CYMometer?

### ➤ 功能描述:

控制系统频率计开关状态。

注意：打开该功能会关闭通道的同步输出。

### ➤ 返回格式:

查询返回系统频率计开关状态，0 表示关闭，1 表示打开。

### ➤ 举例:

:SYSTem:CYMometer ON                    打开系统频率计

:SYSTem:CYMometer?                    查询返回 1

## :SYSTem:CYMometer:FREquency?

### ➤ 命令格式:

:SYSTem:CYMometer:FREquency?

### ➤ 功能描述:

获取频率计的当前测量的频率。

### ➤ 返回格式:

查询返回获取频率计的当前测量的频率，单位 Hz，采样科学计数法返回数据。

### ➤ 举例:

:SYSTem:CYMometer:FREquency?        查询返回 2.000000e +3

## :SYSTem:CYMometer:PERiod?

### ➤ 命令格式:

:SYSTem:CYMometer:PERiod?

### ➤ 功能描述:

获取频率计的当前测量的周期。

### ➤ 返回格式:

查询返回获取频率计的当前测量的周期，单位 S，采样科学计数法返回数据。

### ➤ 举例:

:SYSTem:CYMometer:PERiod?            查询返回 2.000000e -3

## :SYSTem:CYMometer:DUTY?

### ➤ 命令格式:

:SYSTem:CYMometer:DUTY?

### ➤ 功能描述:



获取频率计的当前测量的占空比，以科学计数法返回。

➤ **返回格式：**

查询返回获取频率计的当前测量的占空比，单位%。

➤ **举例：**

:SYSTem:CYMometer:DUTY?                      查询返回 2.000000e+01，表示占空比 20%

## CHANnel 命令

用于设置信号源通道相关功能。

### :CHANnel<n>:MODE

➤ **命令格式：**

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

:CHANnel<n>:MODE?

➤ **功能描述：**

设置指定通道信号模式，分别为 CONTinue、MODulation、SWEep、BURSt。

<n>: 通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道信号模式。

➤ **举例：**

:CHANnel1:MODE MODulation                      设置通道 1 信号调制输出

:CHANnel1:MODE?                                      查询返回 MODulation

### :CHANnel<n>:PA:OUTPut

➤ **命令格式：**

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

:CHANnel<n>:PA:OUTPut?

➤ **功能描述：**

设置打开或关闭功率放大器的输出。

<n>: 通道号，n 取值 2。

注意：此功能只有在功率放大器模块存在时才起作用。正常的 CH2 和功率放大器输出是互斥的，此时正常的 CH2 不能使用。

➤ **返回格式：**

查询返回功率放大器的输出状态，0 表示关闭，1 表示打开。

➤ **举例：**

:CHANnel2:PA:OUTPut ON                              设置打开功率放大器输出

:CHANnel2:PA:OUTPut?                              查询返回 1

## :CHANnel<n>:OUTPut

### ➤ 命令格式:

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

```
:CHANnel<n>:OUTPut?
```

### ➤ 功能描述:

设置打开或关闭指定通道的输出。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道的输出状态, 0 表示关闭, 1 表示打开。

### ➤ 举例:

```
:CHANnel1:OUTPut ON           设置打开通道 1 输出
```

```
:CHANnel1:OUTPut?           查询返回 1
```

## :CHANnel<n>:INVersion

### ➤ 命令格式:

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

```
:CHANnel<n>:INVersion?
```

### ➤ 功能描述:

设置打开或关闭指定通道反向。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道的反向状态, 0 表示关闭, 1 表示打开。

### ➤ 举例:

```
:CHANnel1:INVersion ON       设置打开通道 1 反向输出
```

```
:CHANnel1:INVersion?       查询返回 1
```

## :CHANnel<n>:OUTPut:SYNC

### ➤ 命令格式:

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

```
:CHANnel<n>:OUTPut:SYNC?
```

### ➤ 功能描述:

设置通道同步输出状态。

注意: 设备只有一个同步输出接口, 同时只能打开一个通道的同步输出。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道的同步输出状态, 0 表示关闭, 1 表示打开。

### ➤ 举例:

:CHANnel1:OUTPut:SYNC ON            设置打开通道 1 同步输出  
:CHANnel1:OUTPut:SYNC?            查询返回 1

#### :CHANnel<n>:LIMit:ENABle

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道限幅开关。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的限幅状态。

➤ **举例:**

:CHANnel1:LIMit:ENABle ON            设置打开通道 1 限幅  
:CHANnel1:LIMit:ENABle?            查询返回 1

#### :CHANnel<n>:LIMit:LOWer

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道限幅下限值。

<voltage>表示电压, 单位当前通道指定单位。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的限幅下限值, 采用科学计数法返回。

➤ **举例:**

:CHANnel1:LIMit:LOWer 2            设置通道 1 限幅下限 2V  
:CHANnel1:LIMit:LOWer?            查询返回 2e+0

#### :CHANnel<n>:LIMit:UPPer

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道限幅上限值。

<voltage>表示电压, 单位当前通道指定单位。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的限幅上限值, 采用科学计数法返回。

➤ **举例:**

:CHANnel1:LIMit:UPPer 2                    设置通道 1 限幅上限 2V  
:CHANnel1:LIMit:UPPer?                    查询返回 2e+0

**:CHANnel<n>:AMPLitude:UNIT**

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道输出幅度单位, 当系统处于高低电平模式时, 设置无用。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的输出幅度单位。

➤ **举例:**

:CHANnel1:AMPLitude:UNIT VPP            设置通道 1 输出幅度单位为 VPP  
:CHANnel1:AMPLitude:UNIT?            查询返回 VPP

**:CHANnel<n>:LOAD**

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道输出负载。

<resistance>表示负载电阻值, 单位为  $\Omega$

<n>: 通道号, n 取值 1、2。

注意: 阻值取值范围为 1~10000, 其中 10000 对应于高阻。

➤ **返回格式:**

查询返回指定通道的负载阻值, 采用科学计数法返回。

➤ **举例:**

:CHANnel1:LOAD 50                    设置通道 1 输出负载 50  $\Omega$   
:CHANnel1:LOAD?                    查询返回 5.000000e+01

**:CHANnel<n>:SElect**

➤ **命令格式:**

:CHANnel<n>:SElect

:CHANnel<n>:SElect?

➤ **功能描述:**

用于选择通道。

<n>: {1|2}, 分别表示 {CH1|CH2 }。

➤ **返回格式:**

查询返回 1 或 0, 分别代表 ON 或 OFF。

➤ **举例:**

:CHAN1:SElect

选择通道 1。

:CHAN1:SElect?

查询返回 1, 表示通道被选中。

## 连续

### :CHANnel<n>:BASE:WAVe

➤ **命令格式:**

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

:CHANnel<n>:BASE:WAVe?

➤ **功能描述:**

设置指定通道基波类型。分别为正弦波、方波、脉冲波、三角波、任意波、噪声、直流。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的基波类型。

➤ **举例:**

:CHANnel1:BASE:WAVe SIne

设置通道 1 基本类型为正弦波

:CHANnel1:BASE:WAVe?

查询返回 SIne

### :CHANnel<n>:BASE:FREQuency

➤ **命令格式:**

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

:CHANnel<n>:BASE:FREQuency?

➤ **功能描述:**

设置指定通道输出频率。

<freq>表示频率值, 单位 Hz。(1e-6Hz ~ 当前波形允许最大频率)

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道的输出频率, 采用科学计数法返回。

➤ **举例：**

:CHANnel1:BASE:FREQuency 2000                    设置通道 1 输出频率 2KHz  
:CHANnel1:BASE:FREQuency?                        查询返回 2e+3

**:CHANnel<n>:BASE:PERiod**

➤ **命令格式：**

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

➤ **功能描述：**

设置指定通道输出周期。

<period>表示周期，单位 S。

若为正弦波：范围为（当前允许最大时间 ~ 1e6s）

<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道的限幅上限值，采用科学计数法返回。

➤ **举例：**

:CHANnel1:BASE:PERiod 0.002                    设置通道 1 输出周期 2ms  
:CHANnel1:BASE:PERiod?                        查询返回 2e-3

**:CHANnel<n>:BASE:PHASe**

➤ **命令格式：**

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

➤ **功能描述：**

设置指定通道输出相位。

<phase>表示相位，单位°，范围-360~360。

<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道的输出相位。

➤ **举例：**

:CHANnel1:BASE:PHASe 20                        设置通道 1 输出相位为 20°  
:CHANnel1:BASE:PHASe?                        查询返回 20

**:CHANnel<n>:BASE:AMPLitude**

➤ **命令格式：**



<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号输出高值, 采用科学计数法返回。

➤ **举例:**

```
:CHANnel1:BASE:HIGH 2          设置通道 1 信号输出高值为 2V
:CHANnel1:BASE:HIGH?          查询返回 2e+0
```

**:CHANnel<n>:BASE:LOW**

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道信号输出低值。

<voltage>表示电压, 单位当前通道指定单位。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号输出低值, 采用科学计数法返回。

➤ **举例:**

```
:CHANnel1:BASE:LOW 2          设置通道 1 信号输出低值为 2V
:CHANnel1:BASE:LOW?          查询返回 2e+0
```

**:CHANnel<n>:BASE:DUTY**

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道信号输出占空比。

<duty>表示占空比, 单位%, 范围 0~100。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号输出占空比。

➤ **举例:**

```
:CHANnel1:BASE:DUTY 20        设置通道 1 信号输出占空比为 20%
:CHANnel1:BASE:DUTY?        查询返回 20
```



### :CHANnel<n>:BASE:ARB

➤ **命令格式:**

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

:CHANnel<n>:BASE:ARB?

➤ **功能描述:**

设置指定通道加载基波任意波源下某文件任意波形数据。

<n>: 通道号, n 取值 1、2。

<source>: {INTernal|EXTernal }, 分别内部、外部两种。

<filename>: 任意波文件名称。

➤ **举例:**

:CHANnel1:BASE:ARB INTernal, Common/AbsSine.bsv

### :CHANnel<n>:RAMP:SYMMetry

➤ **命令格式:**

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

:CHANnel<n>:RAMP:SYMMetry?

➤ **功能描述:**

设置指定通道斜波信号输出对称度。

< symmetry >表示对称度, 单位%, 范围 0~100。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道斜波信号输出对称度, 以科学计数法返回。

➤ **举例:**

:CHANnel1:RAMP:SYMMetry 20                                    设置通道 1 斜波信号对称度为 20%

:CHANnel1:RAMP:SYMMetry?                                    查询返回 2.000000e+01

### :CHANnel<n>:PULSe:RISe

➤ **命令格式:**

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

:CHANnel<n>:PULSe:RISe?

➤ **功能描述:**

设置指定通道信号脉冲波上升沿脉宽。

<width>表示脉宽, 单位 S。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号脉冲波上升沿脉宽, 采用科学计数法返回。

➤ **举例:**

:CHANnel1:PULSe:RISe 0.002                    设置通道 1 信号上升沿脉宽为 2ms  
:CHANnel1:PULSe:RISe?                        查询返回 2e-3

#### :CHANnel<n>:PULSe:FALL

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道信号脉冲波下降沿脉宽。

<width>表示脉宽，单位 S。

<n>: 通道号，n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号脉冲波下降沿脉宽，采用科学计数法返回。

➤ **举例:**

:CHANnel1:PULSe:FALL 0.002                    设置通道 1 信号下降沿脉宽为 2ms  
:CHANnel1:PULSe:FALL?                        查询返回 2e-3

## 调制

#### :CHANnel<n>:MODulate:TYPE

➤ **命令格式:**

:CHANnel<n>:MODulate:TYPE <type>  
:CHANnel<n>:MODulate:TYPE?

➤ **功能描述:**

设置指定通道信号调制类型。

<type>: { AM|FM|PM|ASK|FSK|PSK|PWM }

分别为调幅、调频、调相、幅移键控、频移键控、相移键控、脉宽调制。

<n>: 通道号，n 取值 1、2。

➤ **返回格式:**

查询返回指定通道信号调制类型。

➤ **举例:**

:CHANnel1:MODulate:TYPE AM                    设置通道 1 信号 AM 调制  
:CHANnel1:MODulate:TYPE?                    查询返回 AM

## :CHANnel<n>:MODulate:WAVE

### ➤ 命令格式:

```
:CHANnel<n>:MODulate:WAVE { SINE|SQUare|UPRamp|DNRamp|ARB|NOISe }
```

```
:CHANnel<n>:MODulate:WAVE?
```

### ➤ 功能描述:

设置指定通道信号调制波类型，分别为正弦波、方波、上三角、下三角、任意波、噪声。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道信号调制波类型。

### ➤ 举例:

```
:CHANnel1:MODulate:WAVE SINE          设置通道 1 信号调制波类型为正弦波
```

```
:CHANnel1:MODulate:WAVE?              查询返回 SINE
```

## :CHANnel<n>:MODulate:SOURce

### ➤ 命令格式:

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

```
:CHANnel<n>:MODulate:SOURce?
```

### ➤ 功能描述:

设置指定通道调制源，分别内部、外部两种。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道调制源。

### ➤ 举例:

```
:CHANnel1:MODulate:SOURce INTernal     设置通道一调制源为内部
```

```
:CHANnel1:MODulate:SOURce?            查询返回 INTernal
```

## :CHANnel<n>:MODulate:FREQuency

### ➤ 命令格式:

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

```
:CHANnel<n>:MODulate:FREQuency?
```

### ➤ 功能描述:

设置指定通道信号调制频率。

<freq>表示频率，单位 Hz。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道信号调制频率，返回采样科学计数法表示。

➤ **举例：**

:CHANnel1:MODulate:FREQuency 2000      设置通道 1 信号调制频率 2KHz  
:CHANnel1:MODulate:FREQuency?      查询返回 2e+3

**:CHANnel<n>:MODulate:ARB**

➤ **命令格式：**

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

➤ **功能描述：**

设置指定通道加载调制任意波源下某文件任意波形数据。

<n>：通道号，n 取值 1、2。

<source>：{INTernal|EXTernal }，分别内部、外部两种。

<filename>：任意波文件名称。

➤ **举例：**

:CHANnel1:MODulate:ARB INTernal,Common/AbsSine.bsv

**:CHANnel<n>:MODulate:DEPTH**

➤ **命令格式：**

:CHANnel<n>:MODulate:DEPTH { <depth>  
:CHANnel<n>:MODulate:DEPTH?

➤ **功能描述：**

设置指定通道调制深度。

<depth>表示调制深度，单位%。0% ~ 100%，其中 AM 调制深度为 0% ~ 120%

<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道调制深度，以科学计数法返回。

➤ **举例：**

:CHANnel1:MODulate:DEPTH 50      设置通道一调制深度为 50%  
:CHANnel1:MODulate:DEPTH?      查询返回 5.000000e+01

**:CHANnel<n>:MODulate:RATio**

➤ **命令格式：**

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

➤ **功能描述：**

设置指定通道调制速率值，该指令只对具备速率功能调制类型有效。

<ratio>表示速率，单位为 Hz

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道调制速率值, 采用科学计数法返回。

➤ **举例:**

:CHANnel1:MODulate:RATio 100	设置通道 1 位率 100Hz
:CHANnel1:MODulate:RATio?	查询返回 1e+2

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

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道频率偏差。

<freq>表示频率偏移, 单位 Hz。0Hz ~ 当前基波频率

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道频率偏移, 采用科学计数法返回数据。

➤ **举例:**

:CHANnel1:FM:FREQuency:DEV 2000	设置通道一频率偏移 2KHz
:CHANnel1:FM:FREQuency:DEV?	查询返回 2e+3

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

➤ **命令格式:**

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

➤ **功能描述:**

设置指定通道输出相位偏差。

< phase >表示相位偏移, 单位°, 范围 0~360。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道输出相位偏移。

➤ **举例:**

:CHANnel1:PM:PHASe:DEV 30	设置通道一相位偏移 30°
:CHANnel1:PM:PHASe:DEV?	查询返回 30

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

### > 命令格式:

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

### > 功能描述:

设置指定通道输出脉宽调制下脉宽偏差。

< duty >表示脉宽偏差，单位%，范围 0~100。

<n>: 通道号，n 取值 1、2。

### > 返回格式:

查询返回指定通道脉宽调制下脉宽偏差，以科学计数法返回数据。

### > 举例:

:CHANnel1:PWM:DUTY:DEV 10	设置通道一脉宽偏差 10%
:CHANnel1:PWM:DUTY:DEV?	查询返回 1e+1

## :CHANnel<n>:FSK:FREQuency

### > 命令格式:

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

### > 功能描述:

设置指定通道输出频移键控跳频频率，必须要提前指定调制方式，此指令才可生效。

< freq >表示频率，单位 Hz。

<n>: 通道号，n 取值 1、2。

### > 返回格式:

查询返回指定通道输出跳频频率，以科学计数法返回数据。

### > 举例:

:CHANnel1:FSK:FREQ 2000	设置通道一输出跳频频率 2KHz
:CHANnel1:FSK:FREQ?	查询返回 2e+3

## :CHANnel<n>:PSK:PHASe

### > 命令格式:

:CHANnel<n>:PSK:PHASe { < phase > }  
:CHANnel<n>:PSK:PHASe?

### > 功能描述:

设置指定通道输出相移键控相位值，必须要提前指定调制方式，此指令才可生效。

< phase>表示相位，单位°，范围 0~360。

<n>: 通道号，n 取值 1、2。

### > 返回格式:

查询返回指定通道相移键控相位值，以科学计数法返回数据。

➤ **举例：**

:CHANnel1:PSK:PHAS 90                    设置通道一输出相位 90°  
:CHANnel1:PSK:PHAS?                    查询返回 9e+1

## 扫频

### :CHANnel<n>:SWEep:TYPe

➤ **命令格式：**

:CHANnel<n>:SWEep:TYPe { LINE|LOG }  
:CHANnel<n>:SWEep:TYPe?

➤ **功能描述：**

设置指定通道扫频模式，分别为线性扫频、对数扫频。  
<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道扫频模式。

➤ **举例：**

:CHANnel1:SWEep:TYPe LINE            设置通道一线性扫频模式  
:CHANnel1:SWEep:TYPe?                查询返回 LINE

### :CHANnel<n>:SWEep:FREQuency:STARt

➤ **命令格式：**

:CHANnel<n>:SWEep:FREQuency:STARt <freq>  
:CHANnel<n>:SWEep:FREQuency:STARt?

➤ **功能描述：**

设置指定通道扫频的起始频率。  
<freq>表示频率，单位 Hz。

<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道扫频的起始频率，以科学计数法返回数据。

➤ **举例：**

:CHANnel1:SWE:FREQ:STAR 2000        设置通道一输出扫频的起始频率 2KHz  
:CHANnel1:SWE:FREQ:STAR?            查询返回 2e+3

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

➤ **命令格式：**

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

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

➤ **功能描述:**

设置指定通道扫频的截止频率。

<freq>表示频率, 单位 Hz。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道输出扫频的截止频率, 以科学计数法返回数据。

➤ **举例:**

:CHANnel1:SWE:FREQ:STOP 2000                    设置通道一输出扫频的截止频率 2KHz

:CHANnel1:SWE:FREQ:STOP?                    查询返回 2e+3

:CHANnel<n>:SWEep:TIME

➤ **命令格式:**

:CHANnel<n>:SWEep:TIME <time>

:CHANnel<n>:SWEep:TIME?

➤ **功能描述:**

设置指定通道扫频时的扫描时间。

<time>表示时间, 单位 S。范围为: 1ms ~ 500s

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道扫频时的扫描时间, 以科学计数法返回数据。

➤ **举例:**

:CHANnel1:SWEep:TIME 2                    置通道一扫频时的扫描时间为 2S

:CHANnel1:SWEep:TIME?                    查询返回 2e+0

## 猝发

:CHANnel<n>:BURSt:TYPE

➤ **命令格式:**

:CHANnel<n>:BURSt:TYPE {NCYC|GATe|INFINit}

:CHANnel<n>:BURSt:TYPE?

➤ **功能描述:**

设置指定通道猝发类型, 分别为 N 周期、门控、无限。

<n>: 通道号, n 取值 1、2。

➤ **返回格式:**

查询返回指定通道猝发类型。



➤ **举例：**

:CHANnel1:BURSt:TYPe NCYC                      设置通道一通道 N 周期猝发  
:CHANnel1:BURSt:TYPe?                              查询返回 NCYC

**:CHANnel<n>:BURSt:PERiod**

➤ **命令格式：**

:CHANnel<n>:BURSt:PERiod <period >  
:CHANnel<n>:BURSt:PERiod?

➤ **功能描述：**

设置指定通道猝发周期。  
< period >表示时间，单位 S。  
<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道猝发周期，以科学计数法返回数据。

➤ **举例：**

:CHANnel1:BURSt:PERiod 5ms                      设置通道一猝发周期 5ms  
:CHANnel1:BURSt:PERiod?                              查询返回 5e-3

**:CHANnel<n>:BURSt:PHASe**

➤ **命令格式：**

:CHANnel<n>:BURSt:PHASe <phase>  
:CHANnel<n>:BURSt:PHASe?

➤ **功能描述：**

设置指定通道猝发相位。  
< phase >表示相位，单位°。范围为：0 ~ 360  
<n>：通道号，n 取值 1、2。

➤ **返回格式：**

查询返回指定通道猝发相位，以科学计数法返回数据。

➤ **举例：**

:CHANnel1:BURSt:PHASe 18                              设置通道一猝发相位 18°  
:CHANnel1:BURSt:PHASe?                              查询返回 1.8e+1

**:CHANnel<n>:BURSt:CYCLes**

➤ **命令格式：**

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

➤ **功能描述：**

设置指定通道猝发循环次数。

< cycles >表示循环次数，整型数据。

<n>: 通道号，n 取值 1、2。

➤ **返回格式:**

查询返回指定通道猝发循环次数，以科学计数法返回。

➤ **举例:**

:CHANnel1:BURSt:CYCLes 2                      设置指定通道猝发循环次数为 2

:CHANnel1:BURSt:CYCLes?                      查询返回 2e+0

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

➤ **命令格式:**

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

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

➤ **功能描述:**

设置指定通道门控猝发极性，分别为正极性、负极性。

<n>: 通道号，n 取值 1、2。

➤ **返回格式:**

查询返回指定通道门控猝发极性。

➤ **举例:**

:CHANnel1:BURSt:GATe:POLarity POSitive                      设置通道一门控猝发极性为正极性

:CHANnel1:BURSt:GATe:POLarity?                      查询返回 POSitive

**:CHANnel<n>:TRIGger:SOURce**

➤ **命令格式:**

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

:CHANnel<n>:TRIGger:SOURce?

➤ **功能描述:**

设置指定通道触发源，该指令只对猝发功能有效。

<n>: 通道号，n 取值 1、2。

➤ **返回格式:**

查询返回指定通道触发源。

➤ **举例:**

:CHANnel1:TRIGger:SOURce INTernal                      设置通道 1 内部触发源

:CHANnel1:TRIGger:SOURce?                      查询返回 INTernal

## :CHANnel<n>:BURSt:TRGEde

### ➤ 命令格式:

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

:CHANnel<n>:BURSt:TRGEde?

### ➤ 功能描述:

设置指定通道猝发模式的触发沿, 该指令只对猝发功能的外部触发有效。

<n>: 通道号, n 取值 1、2。

### ➤ 返回格式:

查询返回指定通道触发输出模式。

### ➤ 举例:

:CHANnel1:BURSt:TRGEde RISe          设置通道 1 上升沿触发输出模式

:CHANnel1:BURSt:TRGEde?          查询返回 RISe

## WARB 命令

用于写任意波形文件指令, 包括基本任意波形和调制任意波形写配置。

## :WARB<n>:MODulate

### ➤ 命令格式:

:WARB<n>:MODulate <arb file>

### ➤ 功能描述:

用于写调制任意波形, 波形数据固定 4k 个点, 先发送该指令, 然后发送任意波形文件数据到信号源。

<arb file>表示任意波形文件。

### ➤ 举例:

:WARB1:MODulate "GateVibar.bsv"          写通道一调制任意波形文件

## :WARB<n>:CARRier

### ➤ 命令格式:

:WARB<n>:CARRier <arb file>

### ➤ 功能描述:

用于写基波任意波形, 波形数据固定 4k 个点, 先发送该指令, 然后发送任意波形文件数据到信号源。

<arb file>表示任意波形文件。

### ➤ 举例:

:WARB1:CARRier "GateVibar.bsv"          写通道一基波任意波形文件

## DISPlay 命令

用于获取信号源显示相关信息。

:DISPPlay?

➤ **命令格式:**

:DISPPlay?

➤ **功能描述:**

用于查询信号源当前屏幕的图像数据。

➤ **返回格式:**

查询返回图像数据，返回的数据符合附录 2: [IEEE 488.2 二进制数据格式](#)。

➤ **举例:**

:DISPPlay?    查询返回图像数据

## 编程说明

描述在编程操作过程中可能出现的一些问题及解决方法。当您遇到如下这些问题时，请按照相应的说明进行处理。

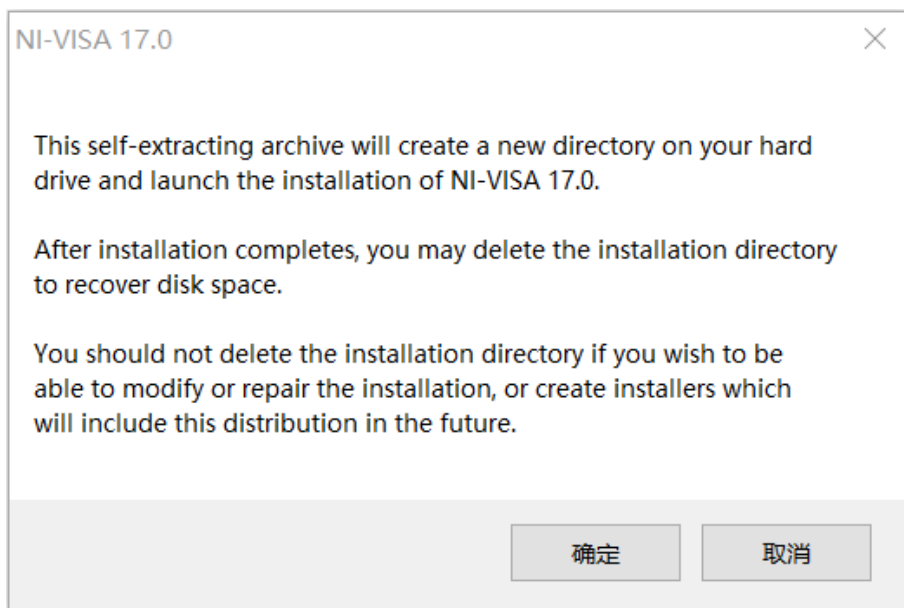
### 编程准备

用户可以通过使用频谱分析仪的 USB 或 LAN 端口，并结合 NI-VISA 和程序语言，远程控制频谱分析仪。编程工作仅适用于在 Windows 操作系统下使用 Visual Studio 和 LabVIEW 开发工具进行编程。

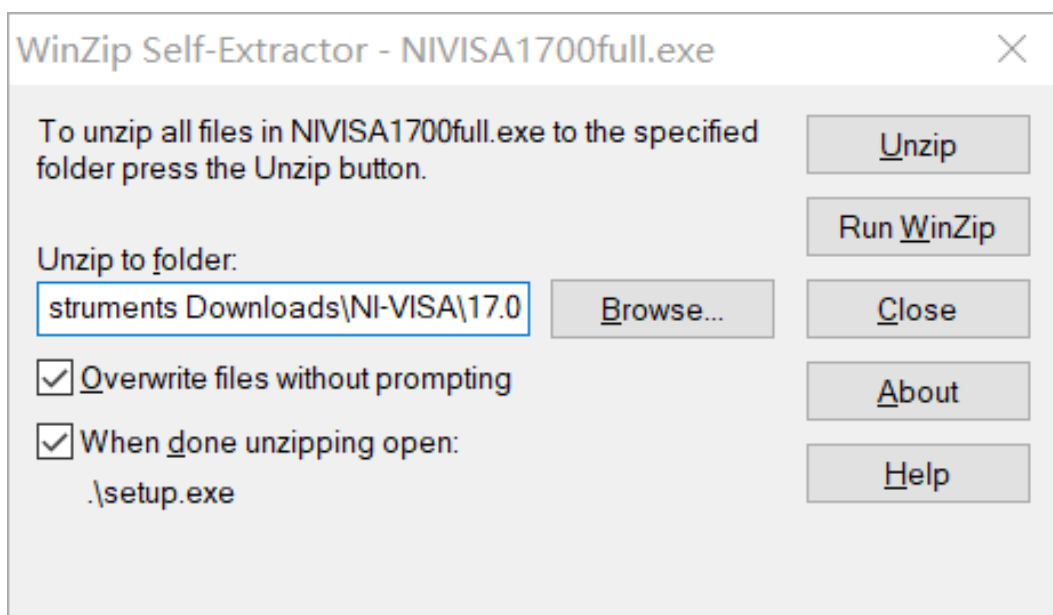
## 1. 建立通信

NI-VISA 是用于计算机与设备之间通信的通信库。NI 软件有两种有效 VISA 安装包：完整版和运行引擎版（Run-Time Engine）。完整版包括 NI 设备驱动和 NI MAX 工具，其中 NI MAX 是用于控制设备的用户界面。虽然驱动和 NI MAX 都很有用，但是它们不用于远程控制。运行引擎版（Run-Time Engine）是一个比完整版更小的文件，它主要用于远程控制。你可以在 NI 官网上下载最新的 NI-VISA 运行引擎或完整版。它们的安装步骤基本相同。按照下列步骤安装 NI-VISA（示例使用 NI-VISA17.0 完整版）：

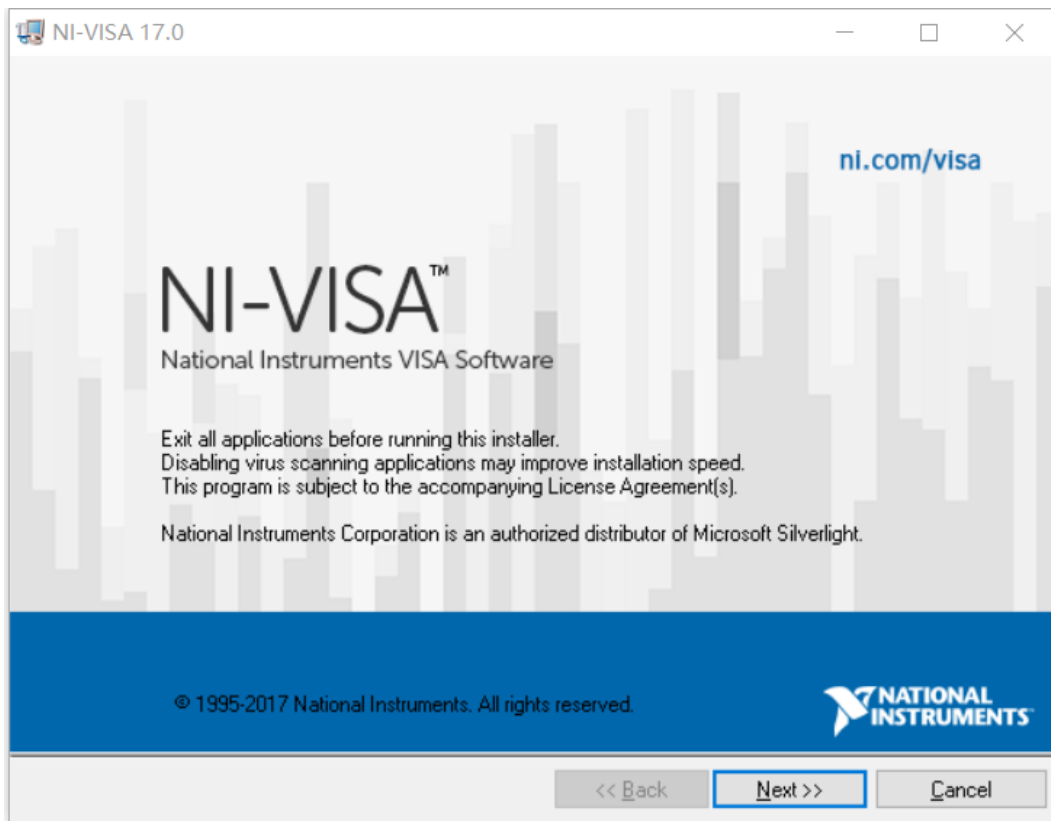
- a. 下载合适版本的 NI-VISA
- b. 双击 NIVISA1700full.exe，弹出对话框如下：



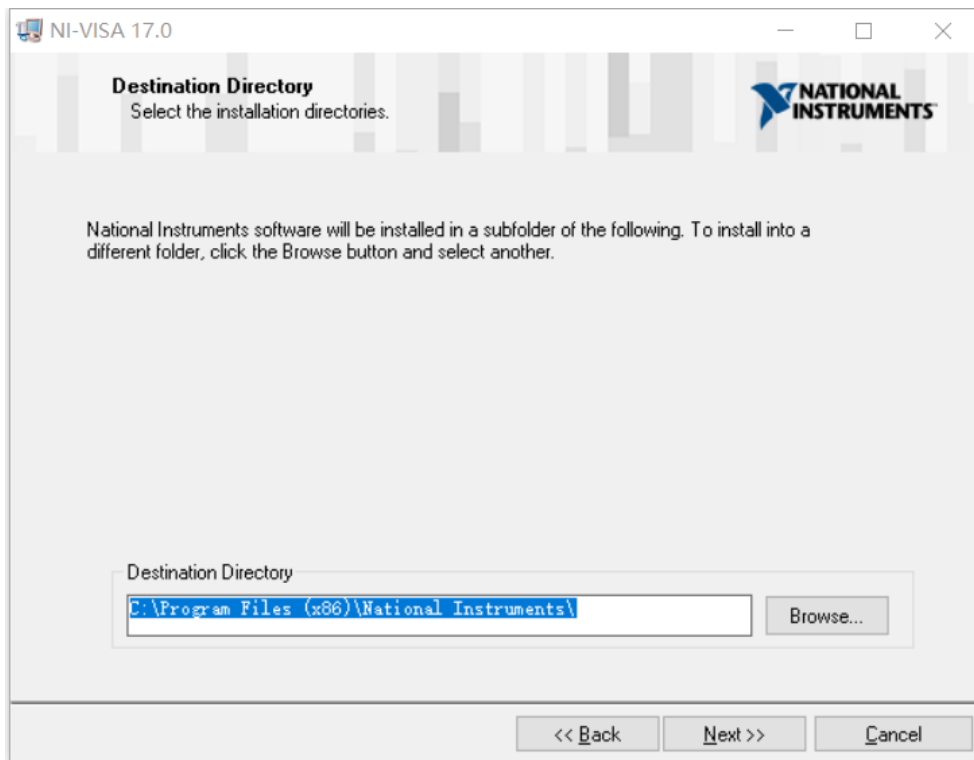
c. 点击确认弹出对话框如下：



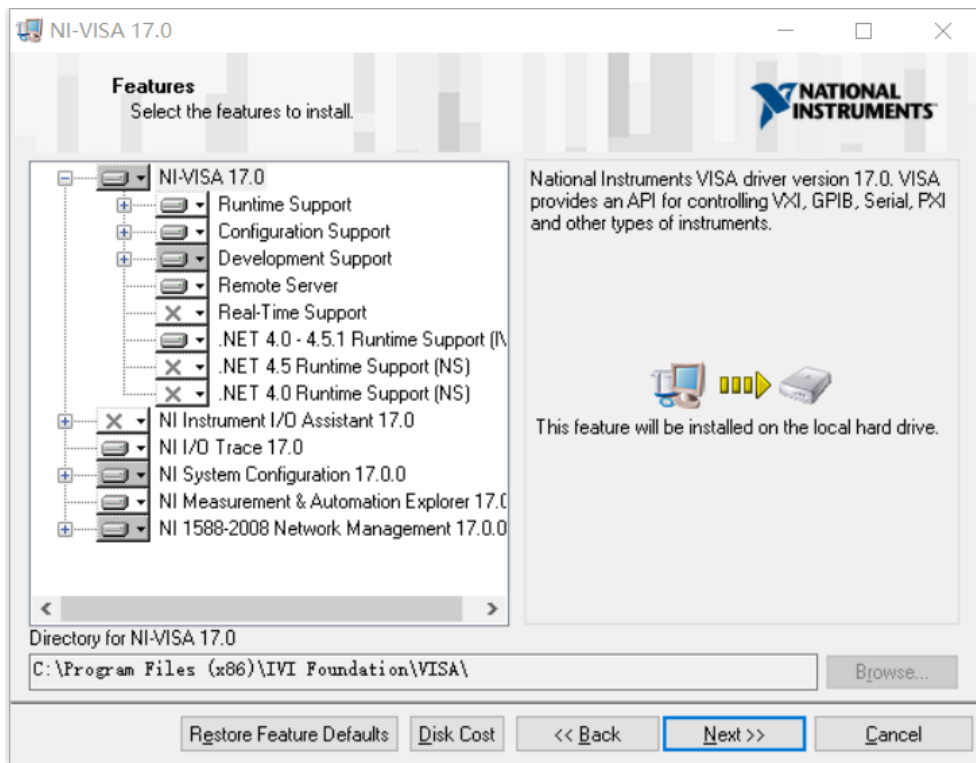
d. 点击 Unzip 解压文件，当解压完成后，安装程序将自动执行。若你的计算机需要安装 .NET Framework4 ，则在安装过程会自动安装 .NET Framework4 。



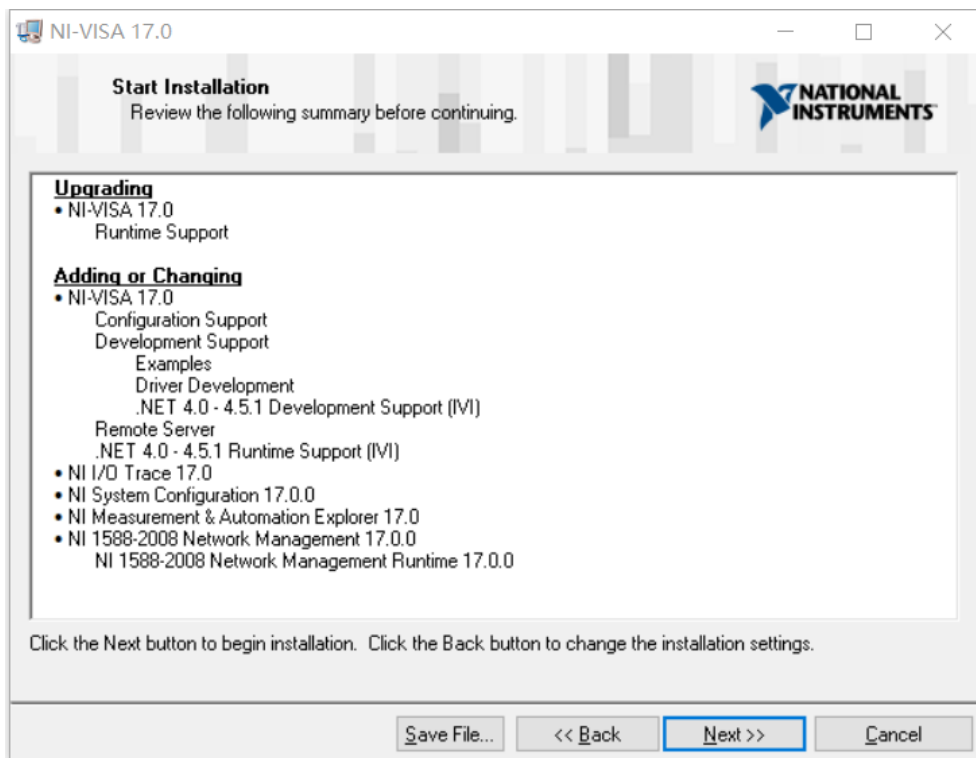
e. NI-VISA 安装对话框如上图所示，点击 Next 开始安装过程。



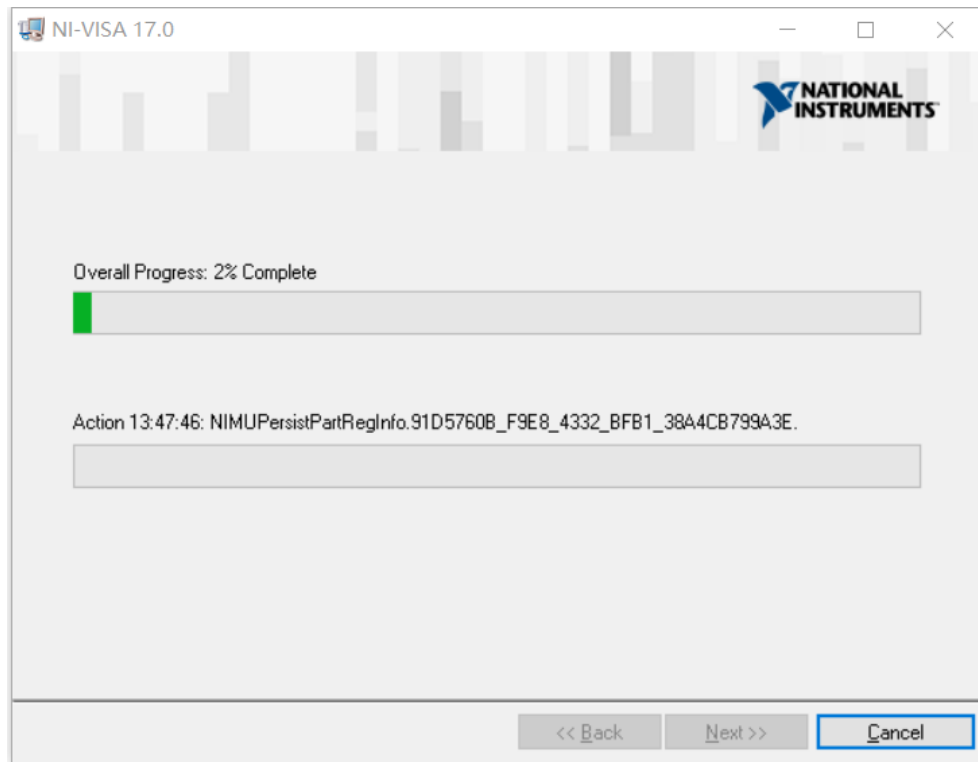
- f. 设置安装路径，默认路径为“C:\Program Files (x86)\National Instruments\”。你也可以修改安装路径。点击 Next ，对话框如下图所示：



- g. 点击 Next 两次，在许可协议对话框下，选择“I accept the above 2 License Agreement (s).” 并点击 Next ，对话框如下图所示：



h. 点击 Next 开始安装：

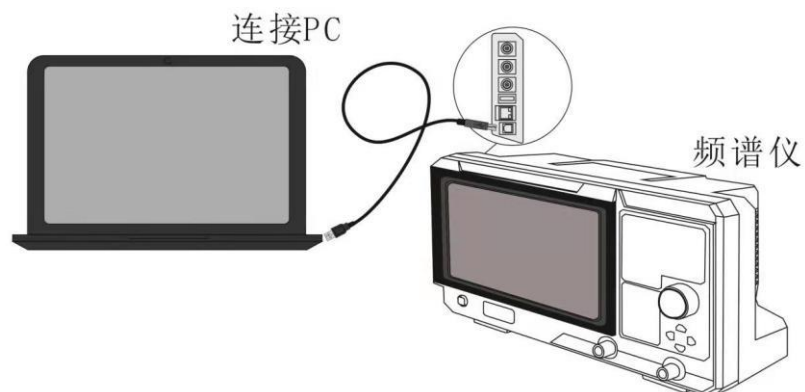


j. 安装完成后，重启电脑

## 2. 连接仪器

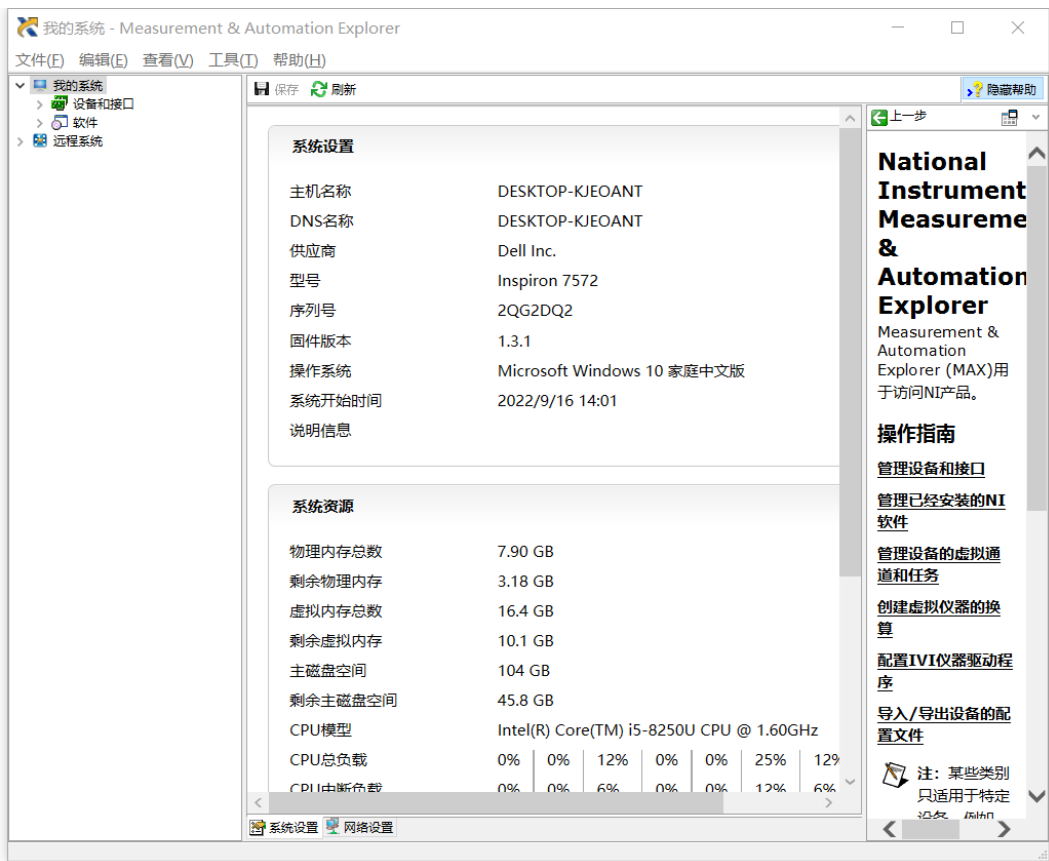
以下通过 USB 方式连接进行介绍。

- 打开频谱分析仪；
- 使用 USB 线将频谱分析仪的 USB Device 端口和计算机的 USB Host 端口连接起来，如下图所示：

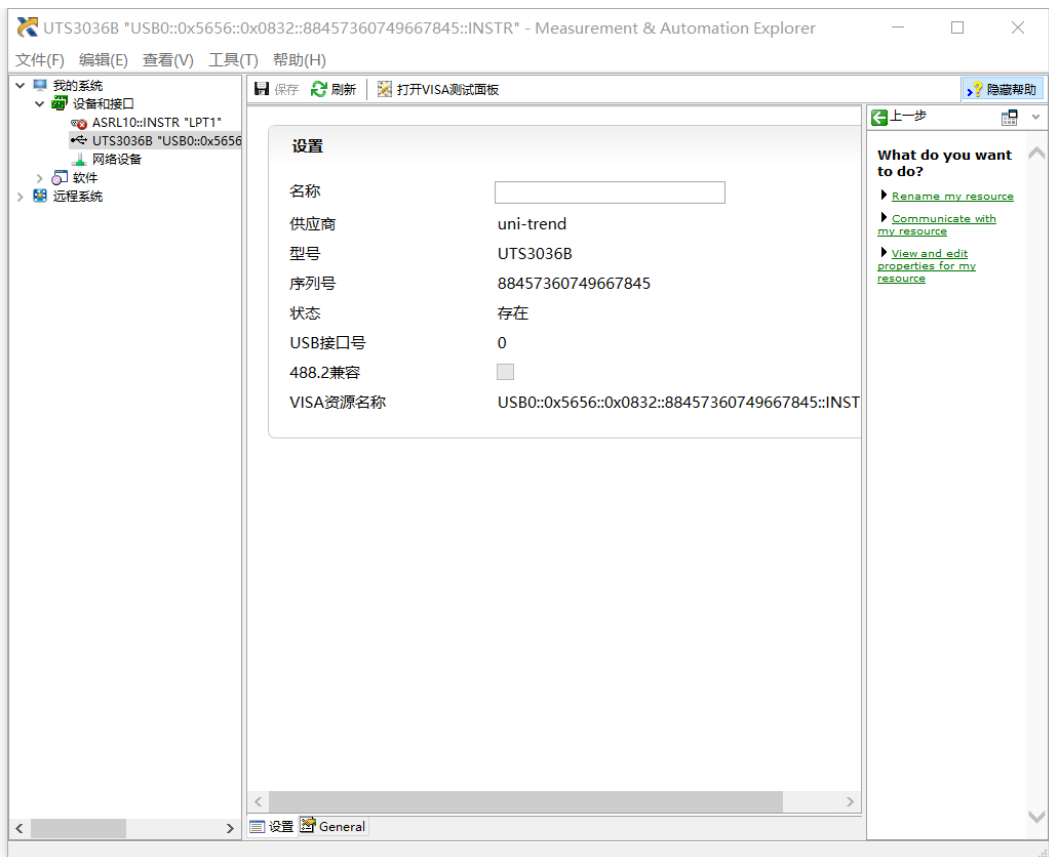


c. 在计算机上打开 NI MAX，弹出如下对话框：

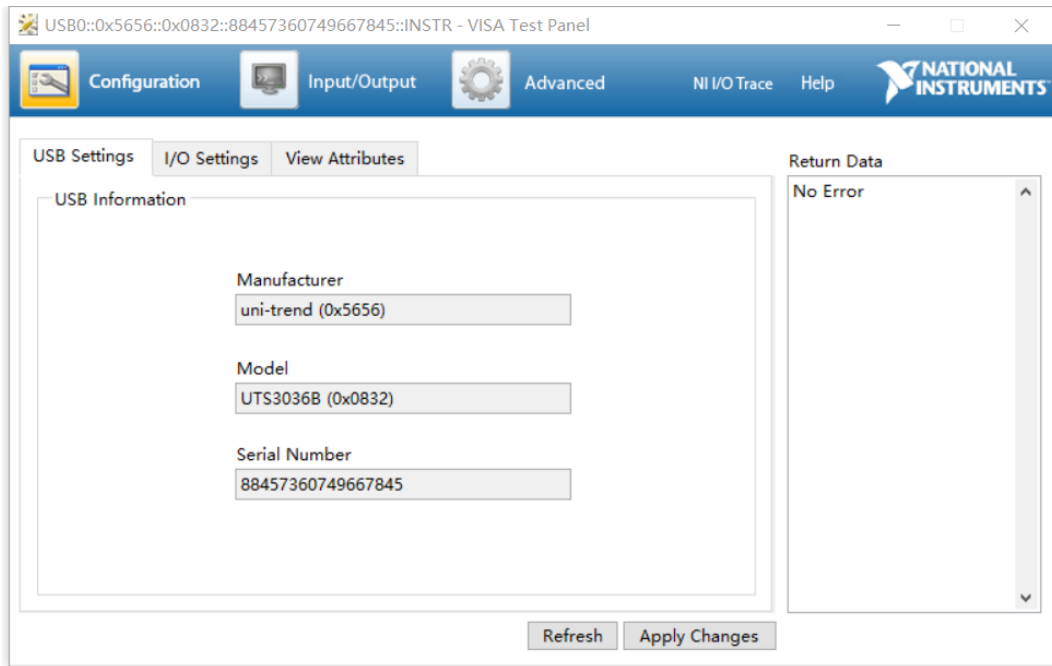




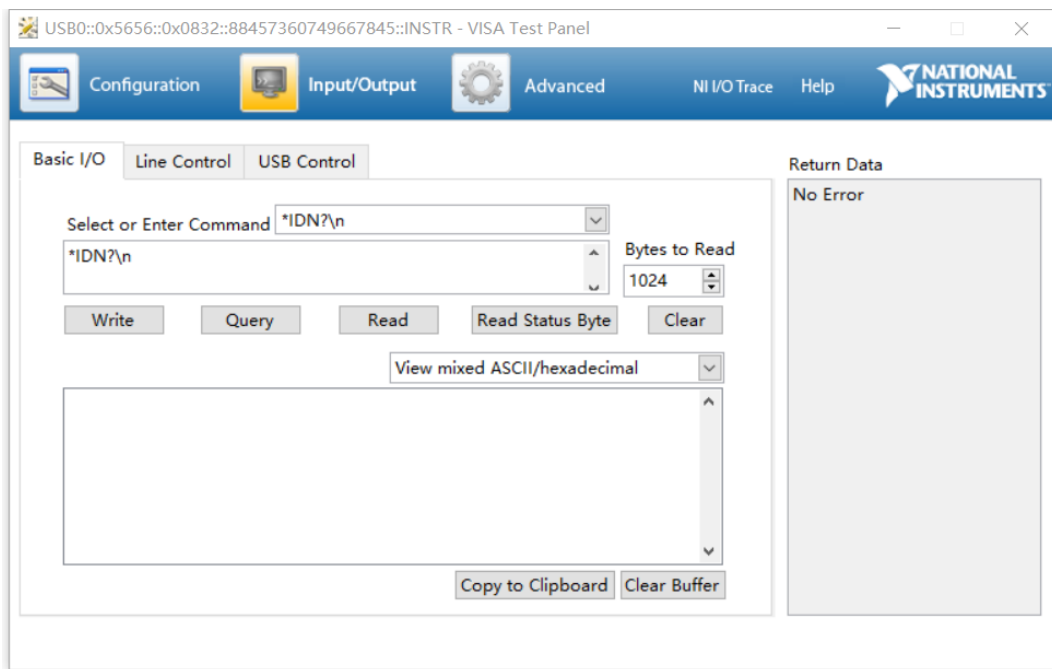
d. 打开设备和接口下拉选项，并选中频谱分析仪驱动，如下图所示：



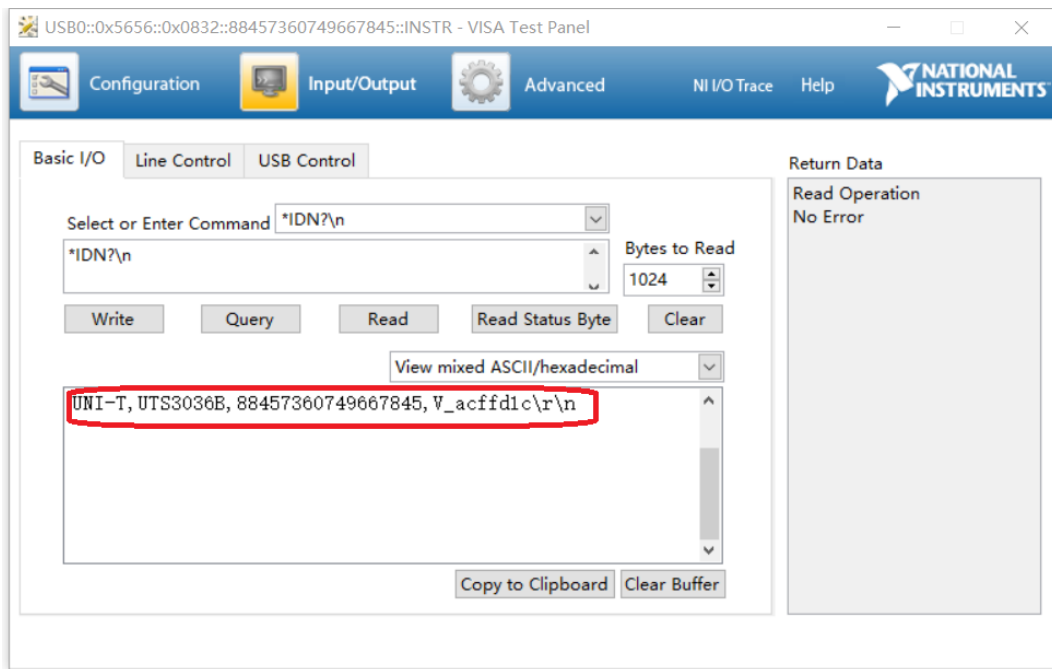
e. 鼠标点击 VISA 测试面板，弹出如下对话框：



f. 鼠标点击 Input/Output 选项，如图下图所示：



g. 鼠标点击 Query 按钮，查询频谱分析仪的 IDN，查询结果如下图红色区域所示：



h. 如果能够查询到频谱分析仪相关信息，就表示频谱分析仪已经可以与计算机通信

## VISA 编程示例

本节给出了一些编程示例。通过这些例子，可以了解如何使用 VISA，并结合编程手册的命令实现对仪器设备的控制。通过下面的例子，你可以开发更多应用。

### VC++ 示例

- 环境：Window 系统，Visual Studio。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备，并在 NI-VISA 上发送 "\*IDN?" 命令来查询设备信息。
- 步骤：

1. 打开 Visual Studio 软件，新建一个 VC++ win32 console project。
2. 设置调用 NI-VISA 库的项目环境，分别为静态库和动态库。

#### a) 静态库：

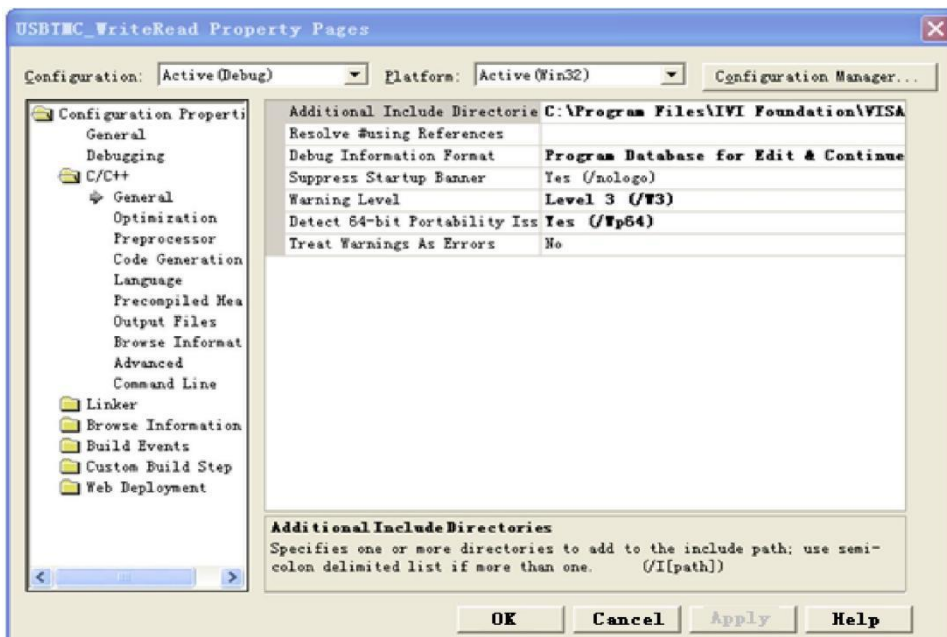
在 NI-VISA 安装路径找：visa.h、visatype.h、visa32.lib 文件，将它们复制到 VC++ 项目的根路径下并添加到项目中。在 projectname.cpp 文件上添加下列两行代码：

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

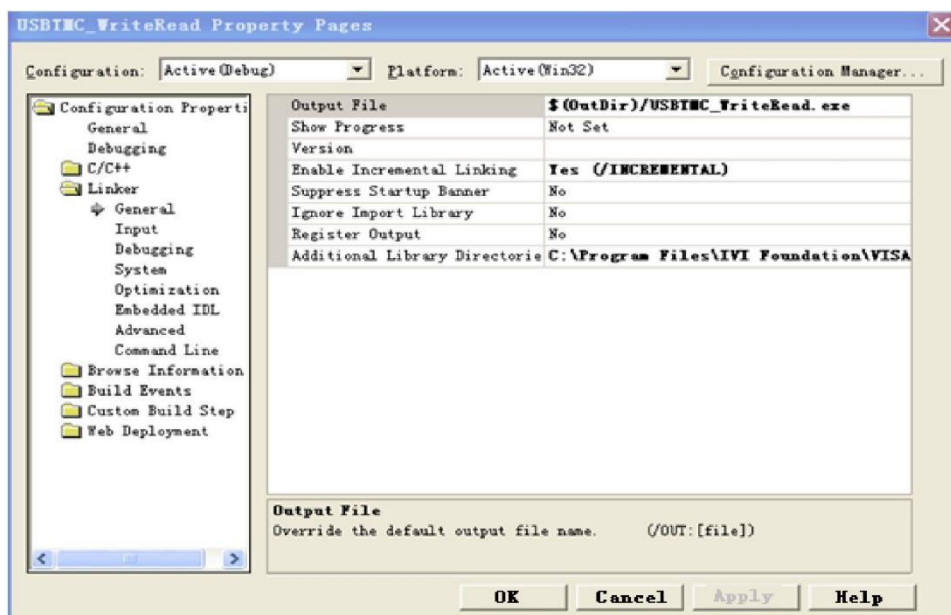
#### b) 动态库：

点击 "project >> properties"，在属性对话框左侧选择 "c/c++ --- General" 中，将 "Additional Include Directories" 项的值设置为 NI-VISA 的安装路径，（例如：C:\ProgramFiles\IVI

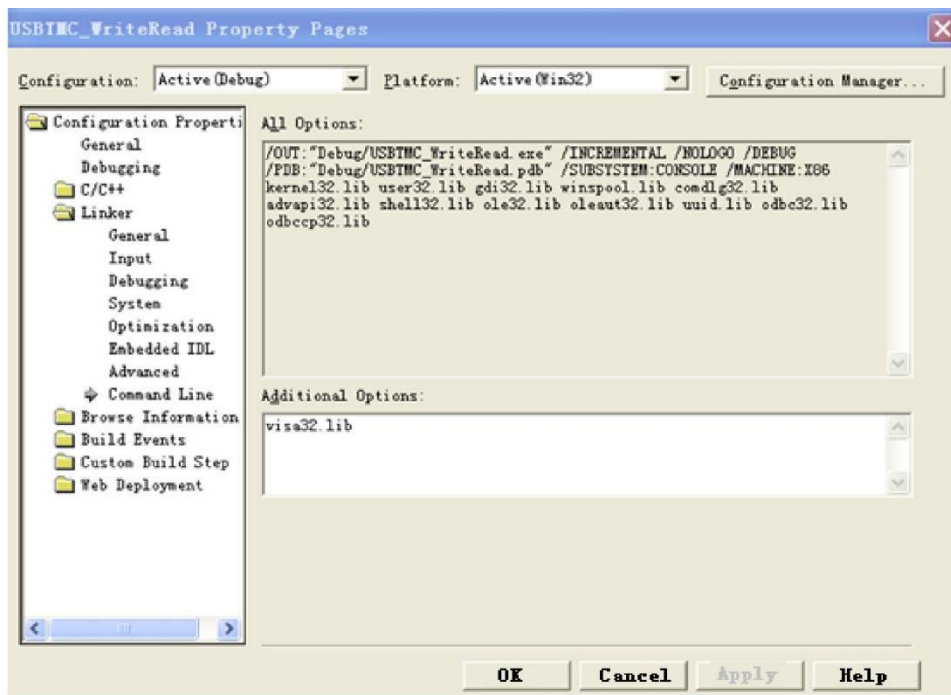
Foundation\VISA\WinNT\include), 如下图所示:



在属性对话框左侧选择"Linker-General", 并将"Additional Library Directories"项的值设置为 NI-VISA 的安装路径, (例如: C:\Program Files\IVI Foundation\VISA\WinNT\include), 如下图所示:



在属性对话框左侧选择"Linker-Command Line", 将"Additional"项的值设置为 visa32.lib, 如下图所示:



在 projectname.cpp 文件上添加 visa.h 文件:

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

## 1. 源码:

### a) USBTMC 示例

```
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_BUFLLEN];
    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 * command = "*IDN?\n";
    status = viPrintf(instr, command);
    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 示例

```

int tcp_ip_test(char *pIP)
{
    char outputBuffer[VI_FIND_BUFLLEN];
    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("\nMessage 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#示例

- 环境：Window 系统， Visual Studio。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备，并在 NI-VISA 上发送"\*IDN?"命令来查询设备信息。
- 步骤：
  1. 打开 Visual Studio 软件，新建一个 C# console project。
  2. 添加 VISA 的 C#引用 Ivi.Visa.dll 和 NationalInstruments.Visa.dll。
  3. 源码：
    - a) USBTMC 示例

```

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 示例

```

class Program
{
    void tcp_ip_test(string ip)
    {
        using (var rmSession = new ResourceManager())
        {
            try
            {
                var resource = string.Format("TCP/IP0::[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");
    }
}

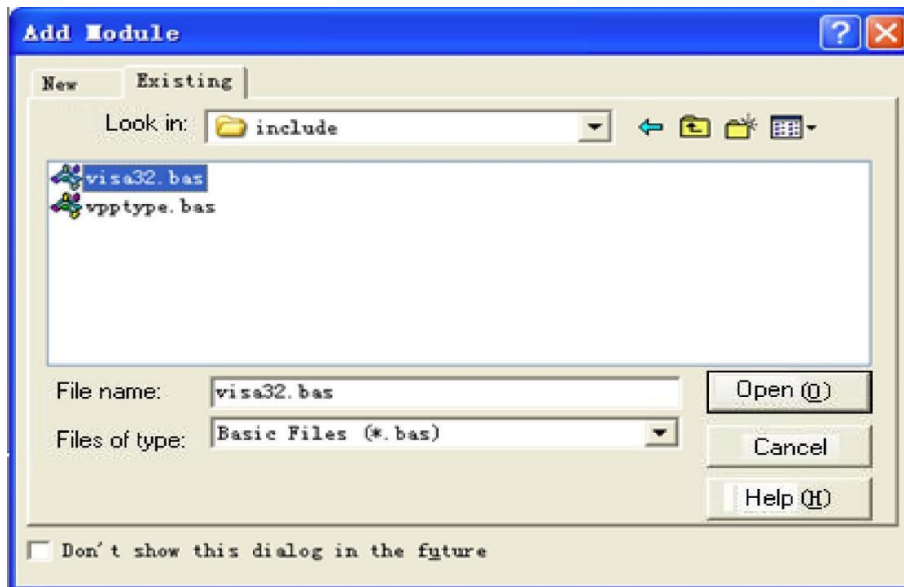
```

## VB 示例

- 环境：Window 系统，Microsoft Visual Basic 6.0。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备，并在 NI-VISA 上发送"\*IDN?"命令来查询设备信息。
- 步骤：

1. 打开 Visual Basic 软件，并新建一个标准的应用程序项目。
2. 设置调用 NI-VISA 库项目环境：点击 Existing tab of Project>>Add

Existing Item, 在 NI-VISA 安装路径下的"include"文件夹中查找 visa32.bas 文件并添加该文件。如下图所示：



3. 源码：

### a) USBTMC 示例

```

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 示例

```

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

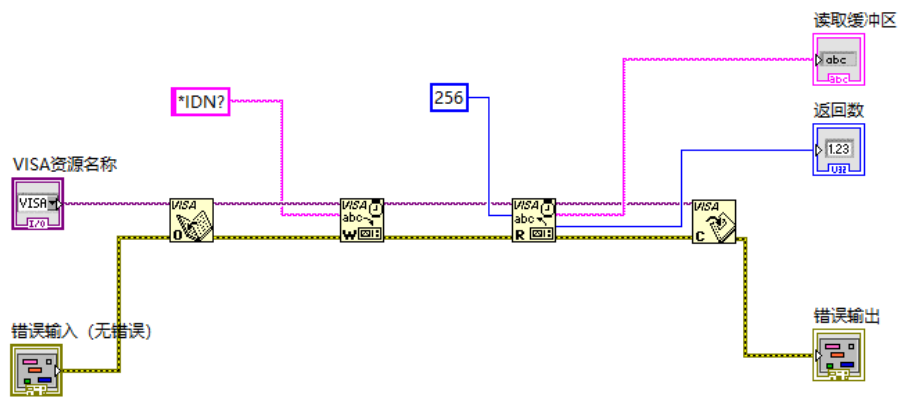
' First we will need to open the default resource manager.
status = viOpenDefaultRM(defaultRM)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Could not open a session to the VISA Resource Manager!"
    tcp_ip_test = status
ExitFunction
EndIf

' 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

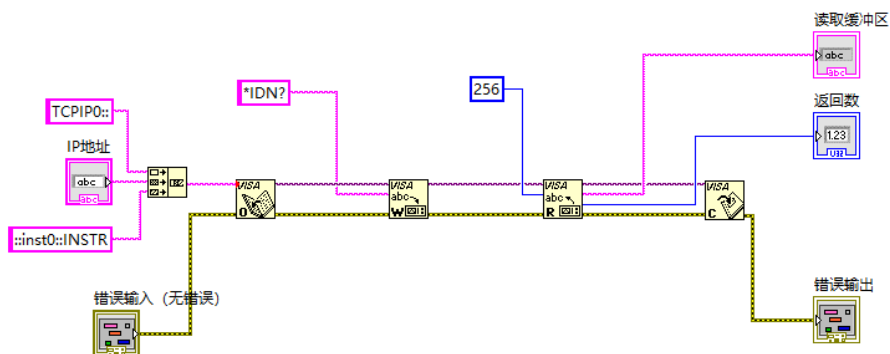
```

## LabVIEW 示例

- 环境：Window 系统, LabVIEW。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备, 并在 NI-VISA 上发送"\*IDN?" 命令来查询设备信息。
- 步骤：
  1. 打开 LabVIEW 软件, 并创建一个 VI 文件。
  2. 添加控件, 右击前面板界面, 从控制列中选择并添加 VISA 资源名、错误输入、错误输出以及部分的指示符。
  3. 打开框图界面, 右击 VISA 资源名称, 并在弹出菜单的 VISA 面板中选择和添加下列功能: VISA Write、VISA Read、VISA Open 和 VISA Close。
  4. VI 打开了一个 USBTMC 设备的 VISA 会话, 并向设备写\*IDN?命令并回读的响应值。当所有通信完成时, VI 将关闭 VISA 会话, 如下图所示:



5. 通过 TCP/IP 与设备通信类似于 USBTMC, 但是你需要将 VISA 写函数和 VISA 读函数设置为同步 I/O, LabVIEW 默认设置为异步 I/O。右键单击节点, 然后从快捷菜单中选择, "Synchronous I/O Mode>>Synchronous" 以实现同步写入或读取数据, 如下图所示:



## MATLAB 示例

- 环境：Window 系统，MATLAB。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备，并在 NI-VISA 上发送"\*IDN?"命令来查询设备信息。
- 步骤：
  1. 打开 MATLAB 软件，点击在 Matlab 界面的 File>>New>>Script 创建一个空的 M 文件。

### 2. 源码：

#### a) USBTMC 示例

```
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 示例

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

- 环境：Window 系统, Python3.8, PyVISA 1.11.0。
- 描述：通过 USBTMC 和 TCP/IP 访问仪器设备，并在 NI-VISA 上发送"\*IDN?"命令来查询设备信息。
- 步骤：
  1. 首先安装 python，然后打开 Python 脚本编译软件，创建一个空的 test.py 文件。
  2. 使用 pip install PyVISA 指令安装 PyVISA，如无法安装，请参考此链接使用说明 (<https://pyvisa.readthedocs.io/en/latest/>)
  3. 源码：

### a) USBTMC 示例

```

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/IP 示例

```

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?'))

```

# 编程应用实例

## 配置正弦波

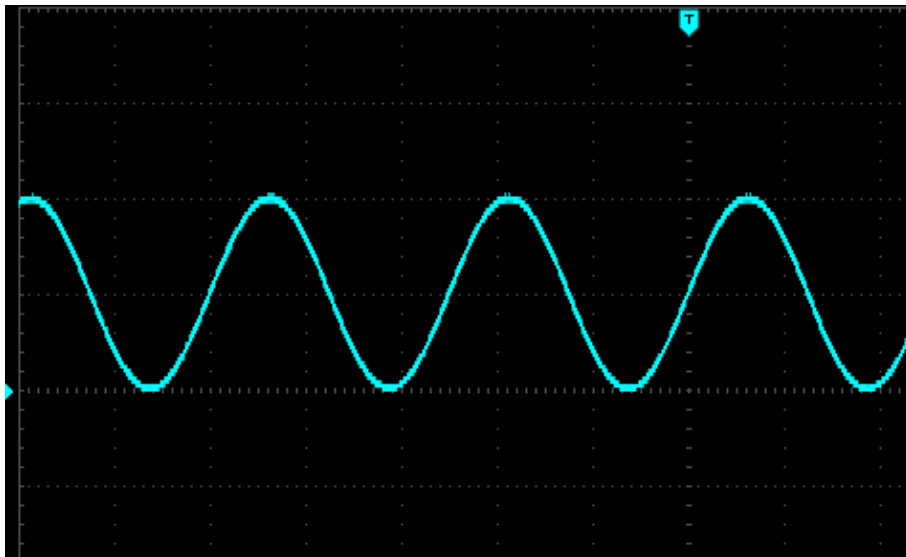
本部分将介绍如何配置正弦波函数。

### 说明

正弦波具有幅度、偏移以及相对于同步脉冲的相位。可使用高电压值和低电压值设置其幅度和偏移。

### 示例

下列波形可由 SCPI 命令系列设置，其中高电平和低电平可用于代替 :CHANnel1:BASE:AMPLitude 和 :CHANnel1:BASE:OFFSet。



以下命令可生成如上所示的正弦波。

```
: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
```

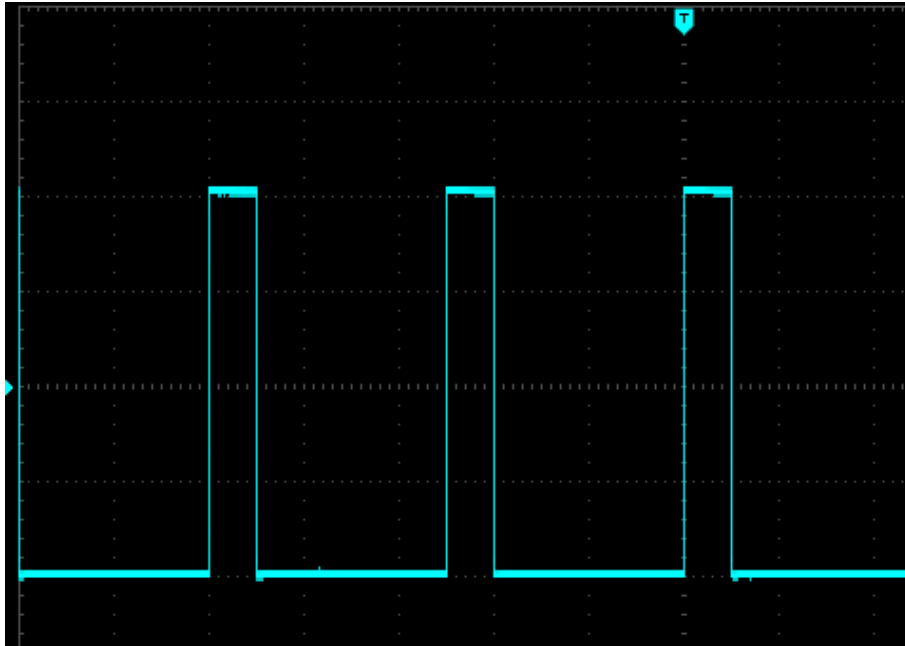
## 配置方波

### 说明

方波具有幅度、偏移以及相对于同步脉冲的相位。它还具有占空比和周期。可使用高电压值和低电压值设置其幅度和偏移。

### 示例

下列波形可由 SCPI 命令系列设置。



以下命令可生成如上所示的方波。

```
: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
```

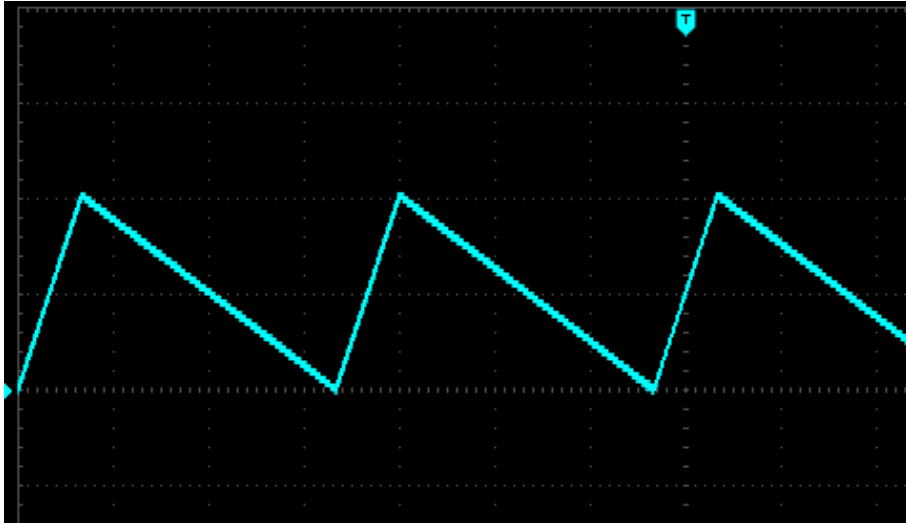
## 配置锯齿波

### 说明

锯齿波具有幅度、偏移以及相对于同步脉冲的相位。它还具有用于创建三角波形和其他类似波形的对称性。可使用高电压值和低电压值设置其幅度和偏移。

### 示例

下列波形可由 SCPI 命令系列设置，其中高电平和低电平可用于代替 :CHANnel1:BASE:AMPLitude 和 :CHANnel1:BASE:OFFSet。



以下命令可生成如上所示的锯齿波。

```
: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
```

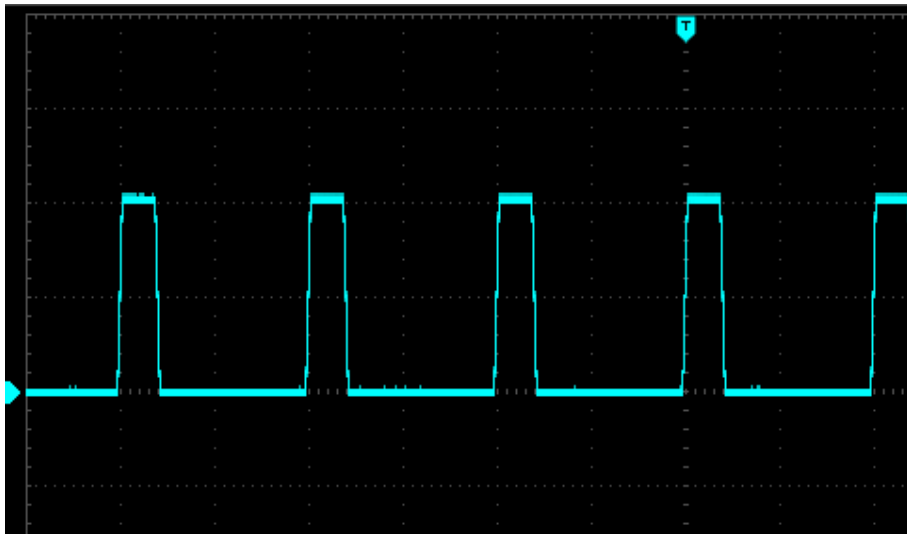
## 配置脉冲波

### 说明

脉冲波具有幅度、偏移以及相对于同步脉冲的相位。它还添加边沿斜率和占空比(或脉冲宽度)。可使用高电压值和低电压值设置其幅度和偏移。

### 示例

下列波形可由 SCPI 命令系列设置，其中高电平和低电平可用于代替 :CHANnel1:BASE:AMPLitude 和 :CHANnel1:BASE:OFFSet。



以下命令可生成如上所示的脉冲波。

```
: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
```

## 配置任意波

本部分将介绍如何配置任意波形。

### 说明

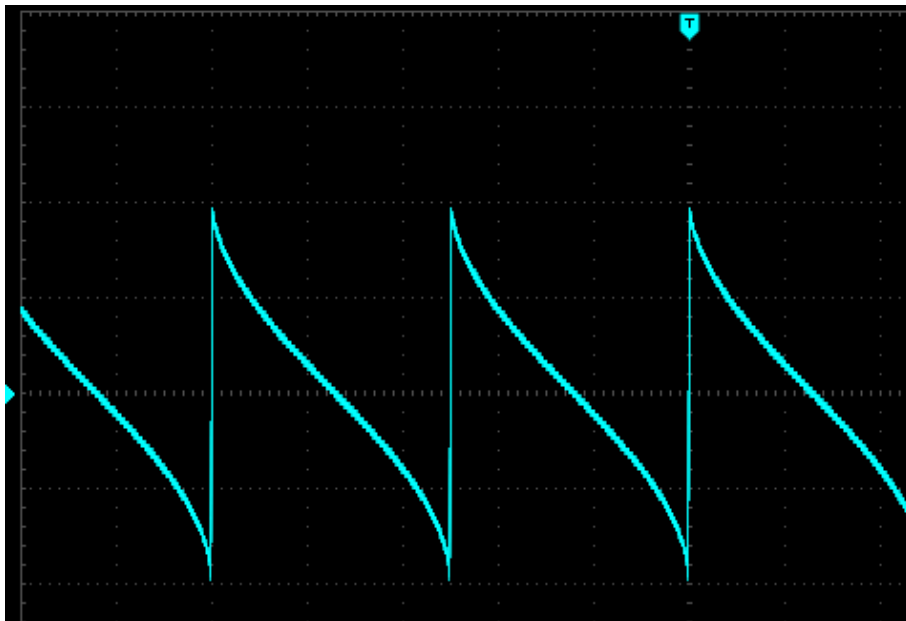
谐波具有频率、幅度、偏移以及相位。它还添加模式、波形文件。

### 示例

下面的代码可加载和修改内置任意波形。

```
: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
```

从这些命令生成的波形如下所示。



## 附录 1：按键列表

按键	功能描述	LED 灯
Wave	波形类型	
Mode	输出模式	√
Utility	系统	
Symbol	数字键符号	
Dot	数字键小数点	
NUM0	数字键 0	
NUM1	数字键 1	
NUM2	数字键 2	
NUM3	数字键 3	
NUM4	数字键 4	
NUM5	数字键 5	
NUM6	数字键 6	
NUM7	数字键 7	
NUM8	数字键 8	
NUM9	数字键 9	
Up	方向键上	
Down	方向键下	
Left	方向键左	
Right	方向键右	
OK	确认键	
CH1	通道一按键	√
CH2	通道二按键	√
F1	选择当前菜单的第一个菜单项	
F2	选择当前菜单的第二个菜单项	
F3	选择当前菜单的第三个菜单项	
F4	选择当前菜单的第四个菜单项	
F5	选择当前菜单的第五个菜单项	
F6	选择当前菜单的第六个菜单项	

## 附录 2：IEEE 488.2 二进制数据格式

DATA 是数据流, 其他为 ASCII 字符, 如下图所示: <#812345678 + DATA + \n>

开 始 符 (1Byte)	长度位宽 (1Byte)	数据总长度 (位宽 Byte)	DATA (n Byte)	结 束 符 (1Byte)
#	x	x x x x x x x x	.....	\n