3212D 毫米波脉冲频率计

程控手册

中电科仪器仪表有限公司

2013 年 10 月

前　言

非常感谢您选择使用中电科仪器仪表有限公司研制生产的 3212D 毫米波脉冲频率计！

我们以最大限度满足您的需求为己任，为您提供高品质的测量仪器，同时带给您一流的售后服务。我们的一贯宗旨是“质量优良，服务周到”，提供满意的产品和服务是我们对用户的承诺，竭诚欢迎您的垂询，联系方式：

电　　话　4000552041

网　　址　http://www.ceyear.com

电子信箱　eiqd@ceyear.com, eibb@ceyear.com

地　　址　山东省青岛经济技术开发区香江路 98 号

邮　　编　266555

本手册是 3212D 毫米波脉冲频率计编程指南。为方便您熟练程控该仪器，请仔细阅读本手册，并正确按照手册指导操作。

由于时间紧迫和笔者水平有限，文字中如有疏漏和不当之处，恳请各位用户批评指正！由于我们的工作失误给您造成的不便我们深表歉意。

本手册是《3212D 毫米波脉冲频率计程控手册》第一版，版本号是 AV2.721.1006SC/1.0

本手册中的内容如有变更，恕不另行通知。

☛　声明：

编者

环境、安全说明

一、 安全保护

1、 仪器安全

1） 仪器运输过程请使用指定包装箱，并且搬运过程避免跌落或剧烈碰撞造成仪器损伤。

2） 检查仪器上的电源保险丝是否正常，否则加电会造成仪器损坏！

3） 请选用 220V 交流三芯稳压电源为仪器设备供电，防止大功率尖峰脉冲干扰对仪器内部硬件造成毁坏。

4） 保证电源良好接地，接地不良或错误可能导致仪器损坏。

5） 操作仪器和设备时请采取佩戴静电手腕等防静电措施，严防静电对仪器的损害。

6） 严禁在仪器输出端注入直流信号，并防止信号的反向功率大于 0.5 W，否则会引起仪器损坏。

7） 如果仪器使用电池供电或内部有电池，请使用相同类型或推荐相当类型的电池进行替换，否则存在爆炸的危险。

2、 人身安全保护注意事项

1） 搬运仪器及包装箱时请选取合适的搬运工具或者应有两人合力搬移，并轻放，以免仪器跌落造成人身伤害。

2） 保证电源良好接地，接地不良或错误可能造成人身伤害。

3） 如果需要擦拭仪器，请断电操作，防止发生触电危险，可以用干的或稍微湿润的软布擦拭仪器外表，千万不要试图擦拭仪器内部。

4） 微波仪器工作在大功率状态下时存在微波辐射的潜在危险，请相应采取防辐射措施。

二、 环境保护

1、 包装箱的处理

我单位承诺产品包装物为无害废弃物，请保留好包装箱和衬垫，以备将来需要运输时使用，也可以按照当地环境法规要求处理产生的包装物。

2、 报废处理

1） 仪器在维修及升级过程中更换下来的零部件由中电科仪器仪表有限公司集中回收处理；仪器报废后禁止随意丢弃或处置，请通知中电科仪器仪表有限公司或交由具有资质的专业回收单位进行回收处理。

2） 如果仪器内部有使用电池，请勿随便丢弃更换下来的电池，应按照化学废品单独回收！

除非另有规定，以上操作请按照国家《废弃电器电子产品回收处理管理条例》和当地环境法律法规处置。

目　录

第一章 远程编程基础

本章描述了远程编程的基础知识以及频率计远控端口的配置。

- 编程和软/硬件层
- 频率计远控端口及配置
- I/O 库和编程语言

第一节 编程和软/硬件层

本型号频率计支持的编程接口为：LAN 和 GPIB。这些接口与 I/O 库和编程语言结合使用可以远程控制频率计。图 1-1 以 LAN 为例显示了接口、I/O 库、编程语言和频率计之间的关系。

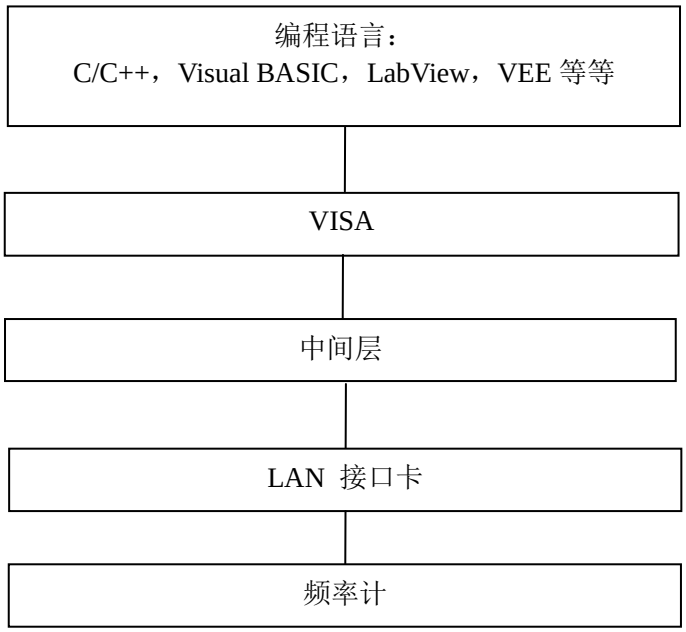


图 1-1 软硬件层

第二节　频率计远控端口及配置

1.2.1GPIB 接口

首先以仪器程控为目的被提出的 GPIB 接口，目前仍被广泛的使用。各种仪器设备通过 GPIB 连接在一起，并且能够被计算机控制。GPIB 及其相关接口操作在 ANSI/IEEE 标准 488.1-1987 和 ANSI/IEEE 标准 488.2-1992 中有详细的定义和描述。关于标准的细节参见 IEEE 网站：http://www.ieee.org

GPIB 以字节为单位来处理信息，数据传输速率能够达到 8MBps，因此 GPIB 的数据传输比较快。GPIB 受设备/系统与计算机之间距离的限制，最大传输电缆总长度 20 米，一般情况下设备间的最大长度不能超过 2 米。

1.2.1.1 设置 GPIB 接口

GPIB 地址的通过菜单【系统】→【程控】→【GPIB 地址】来设置。GPIB 地址的设置范围为 0~30，厂家复位模式下，默认地址为 1。

菜单及操作方法参见图 1-2。

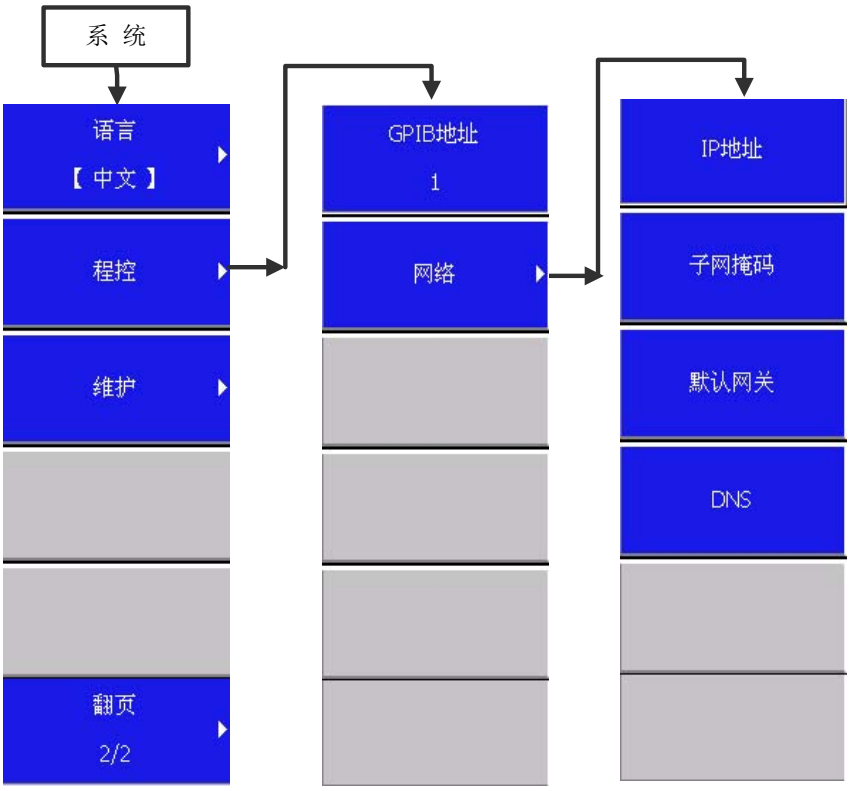


图 1-2 GPIB 接口设置

注意 **当系统中有多于一台频率计时，每台频率计的 GPIB 地址应不同；**

1.2.2 LAN 接口

频率计可通过 10Base-T 和 100Base-T 局域网内计算机进行远程控制，局域网可使各种仪器联系为一体并由网内计算机控制。局域网及其接口操作定义详见 IEEE 802.2 及 http://www.ieee.org.。

由于数据分组传输的原因，LAN 传输数据很快。一般情况下，计算机和频率计之间电缆长度不应超过 100 米（100Base-T 和 10Base-T）。关于 LAN 通信的更多信息参考：http://www.ieee.org。

主控计算机可以使用下面的协议通过 LAN 与频率计通信。

◆ Sockets

1.2.2.1 设置 LAN 接口

通过局域网对频率计进行远程控制时，应保证网络的物理连接畅通。由于不支持 DHCP、域名访问以及广域网络连接，因此频率计的网络程控设置相对简单，通过图 1-3 所示的菜单，将其中“IP 地址”，“子网掩码”，“默认网关”设置到主控制器所在的子网内即可。

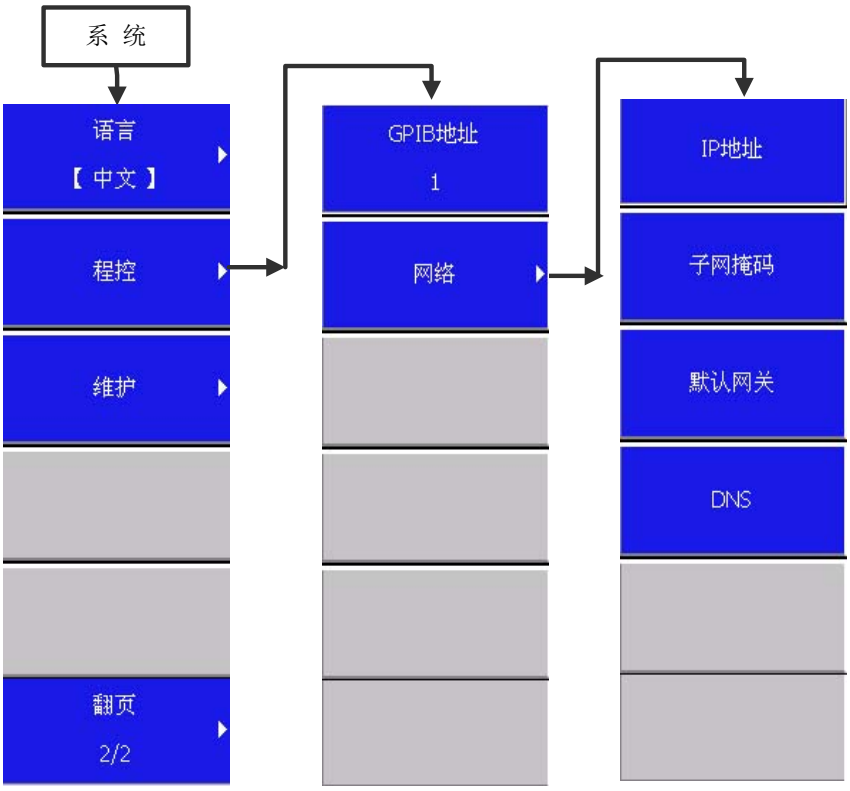


图 1-3 LAN 接口设置

注意 **确保频率计通过 10Base-T LAN 或 100Base-T LAN 电缆物理连接正常。**

本系列频率计只支持单一局域网络控制系统的搭建，且只支持静态 IP 地址的设置，不支持 DHCP，也不支持通过 DNS 和域名服务器访问主机，

1.2.2.2 使用局域网套接字

TCP/IP 协议通过局域网套接字在网络中连接频率计。套接字是计算机网络编程中使用的一个基本方法，它使得使用不同硬件和操作系统的应用程序得以在网络中进行通信。这种方法通过端口（port）使频率计与计算机进行双向通信。

套接字是专门编写的一个软件类，里面定义了 IP 地址、设备端口号等网络通信所必需的信息，整合了网络编程中的一些基本操作。在操作系统中安装了打包的库就可以使用套接字。两个常用的套接字库是 UNIX 中应用的伯克利（Berkeley）套接字库和 Windows 中应用的 Winsock 库。

频率计中的套接字通过应用程序接口（API）兼容 Berkeley socket 和 Winsock。此外，还兼容其他标准套接字 API。频率计可以用 SCPI 命令控制，这些命令由程序中建立的套接字程序发出。

在使用局域网套接字之前，必须选择频率计的套接字端口号，频率计的套接字端口号为 5000。

注意 **前面板的 Type-A 连接器是 USB 主控端口连接器，在 3212D 频率计中，该端口用来连接 USB 1.1 接口的闪存盘，以实现仪器驻机软件的升级，也可以连接 USB 键盘和鼠标对频率**

计进行控制。不能通过该端口来程控仪器。

第三节 I/O 库和编程语言

I/O 库是一些函数集合，这些函数是用来对设备发送命令和接收设备数据的。主控计算机在对频率计进行交互和控制之前必须安装 I/O 库。关于 I/O 库的安装及配置，请参阅您所选择的控制卡及 I/O 库的相关文档资料。

注意 **在使用主控计算机控制频率计之前，请确认您已正确安装且配置必要的端口和 I/O 库。**

Standard Commands for Programming Instructions (SCPI)、IO 库函数和编程语言一起使用，可以远程控制频率计。通常的编程语言包括：

- C/C++
- C#
- HP Basic
- LabView
- Visual Basic® （Visual Basic 是 Microsoft 公司的注册商标）
- Agilent VEE

第二章 SCPI 基础

本章描述了 SCPI 消息是如何组织和表达的，提供了一个 SCPI 语言的概览。

- SCPI 简介
- SCPI 通用术语
- SCPI 命令语法
- SCPI 命令类型
- SCPI 命令树
- SCPI 命令参数和响应
- SCPI 程控消息
- SCPI 命令中数值的进制

第一节 SCPI 简介

定义的标准化的 SCPI 仪器程控消息、响应消息、状态报告结构和数据格式的使用只与仪器测试功能及其性能、精度相关，而不考虑仪器硬件组成、制造厂家、通信物理连接硬件环境和测试程序编制环境。

（1）、程控命令面向测试功能（信号），而不是描述仪器操作。

（2）、减少类似测试功能的控制方法是保证编程相容性的关键。

（3）、在与通信物理层硬件无关的高层次上定义程控消息。

（4）、与编程手段和程序语言无关，SCPI 测试程序容易移植。

（5）、具有可缩性，可适应不同规模的测量控制。

（6）、SCPI 的可扩展性，使其成为“活”标准。

第二节 SCPI 通用术语

下面这些术语贯穿了这部分的余下部分。为了更好的理解以后章节的内容，您需要了解这些术语的确切定义。

控制器

控制器是任何用来与 SCPI 设备通讯的计算机。控制器可能是个人电脑、小型计算机或者卡笼上的插卡。一些人工智能的设备也可作为控制器使用。

设备

设备是任何支持 SCPI 的装置。大部分的设备是电子测量或者激励设备，并使用 GPIB 接口通讯。

程控消息

程控消息是一个或者多个正确格式化过的 SCPI 命令的组合。程控消息告诉设备怎样去测量和输出信号。

响应消息

响应消息是指定 SCPI 格式的数据集合。响应消息总是从设备到控制器或者侦听设备。响应消息告诉控制器关于设备的内部状态或测量值。

命令

命令是指满足 SCPI 标准的指令。控制设备命令的组合形成消息。通常来说，命令包括关键字、参数和标点符号。

事件命令

一些命令是事件，不能被查询。一个事件没有相应的设置，它在一个特定的时刻触发一个动作。

查询

查询是一种特殊类型的命令。查询控制设备去产生适合控制器要求的响应消息。查询语句总是以问号结束。

第三节 SCPI 命令语法

一个典型的命令是由前缀为冒号的关键字构成，关键字后面跟着参数。

语法表达式的惯例见表 2-1，2-2。

表 2-1 命令语法中的特殊字符

符号	含义	举例
|	关键字或参数之间的竖号意味着另外的选择。	:CONFigure:MEASure:MODE CW|PULse 在发送命令时可选择 CW 或 PULse
[]	方括号表示附上的关键字或者参数在构成命令时是可选择的。这些暗含的关键字或者参数甚至在它们被忽略时也将被执行。	[:SENSe]:FREQuency:RESolution? SENSe 是可有可无的项
<>	围绕一个单词的尖括号表示他们在命令中并不是按照字面的含义使用。它们代表需要的项。	:CONFigure:USER:FREQ:STOP <val><unit> 在这个命令中，<val>和<unit>必须用真实的频率值和单位代替

表 2-2 命令语法

字符，关键字和语法	举例
大写的字符代表执行命令所需要的最小字符集合。	[:SENSe]:AVERage:COUNt?, SENS 是最少的需要的部分。
命令的小写字符部分是可选择的；它可以包含在命令的大写部分也可以被忽略。这种灵活性的格式被称为“灵活地听”。更多信息请参照“命令的参数和响应”部分。	:FREQuency :FREQ,:FREQuency 其中任意一个都是正确的。
当一个冒号在两个命令助记符之间，它将命令树中的当前路径下移一层。更多消息请参照“命令树”的命令路径部分。	:TRIGger:MODE? TRIGger 是这个命令的最顶层关键字
空白字符，例如<space>或者<tab>，只要不出现在关键字之间或者关键字之中，通常是被忽略的。 然而，你必须用空白字符将命令和参数分隔开来，这个并不影响当前路径。	:CONF:USER:FREQ:STOP4GHz 是不允许的。 在: STOP 和 4GHz 之间必须要由一个空格。 :CONF:USER:FREQ:STOP 4GHz

第四节 SCPI 命令类型

命令可以分为 2 种类型：通用命令和分系统命令。图 2-1 显示了 2 种命令的差异。通用命令由 IEEE 488.2 定义，用来管理宏、状态寄存器、同步和数据存储。他们都由一个星号开始，因此很容易认出来。例如*IDN？、*OPC、*RST 都是通用命令。通用命令不属于任何分系统，宽带微波毫米波脉冲频率计用同一种方式解译它们而不用考虑当前的路径设置。分系统命令因为有分号（:）而显得很容易辨认。分号用在命令表达式的开头和关键字的中间，每一个命令分系统都是概略对应着宽带微波毫米波脉冲频率计内部一个功能模块的命令集合。例如，系统分系统（:SYSTem）包含系统设置的命令，而状态分系统（:STATus）包含控制状态寄存器的命令。

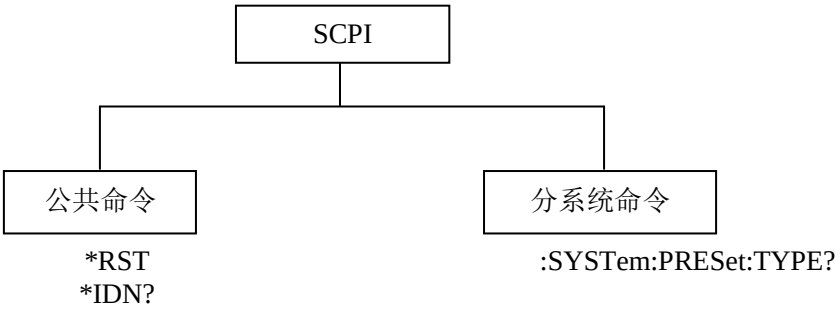


图 2-1 SCPI 命令类型

第五节 SCPI 命令树

大部分编程任务会涉及到分系统命令。在大多数计算机中，SCPI 对分系统命令使用一个相似于文件系统的结构。在 SCPI 中，这种命令结构被称为命令树，如图 2-2 所示：

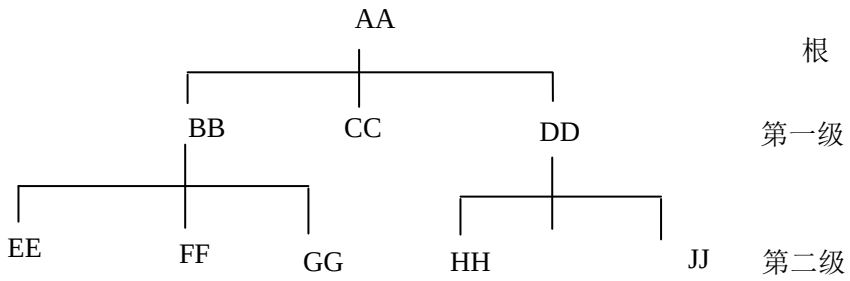


图 2-2 简化的命令树

最靠近顶端的命令是根命令，简单地说就是“根”。你必须根据一个特定的路径到达下一层命令。在下面的例子中，：CONFigure 代表 AA，：MEASure 代表 BB，：MODE 代表 GG。整个命令路径是:CONFigure:MEASure:MODE？（:AA:BB:GG）。

为了在命令树中按照不同的路径访问命令，您必须明白设备怎样解释命令。设备软件中的一个特殊部分——***解释器***，负责解码每一个送到设备的消息。编译器利用一系列的辨认命令树路径的规则，把消息分成单独的命令元。在你发送下一个命令时，编译器保持跟踪***当前路径***，即在命令树中对应的级别。这是十分重要的，因为同样的命令关键字可能出现在不同的路径中，而你使用的特定路径决定了命令关键字的解释结果。

在开机或在*RST（复位）后，置当前路径为根。

第六节 SCPI 命令参数和响应

SCPI 定义了不同的数据格式在程控和响应消息的使用中以符合“***灵活地听***”和“***精确地讲***”的原则。更多的信息请参照 IEEE488.2。“***灵活地听***”指的是命令和参数的格式是灵活的。

例如命令:CONFigure:CARRier:FREQuency:STATe ON|OFF|1|0，宽带微波毫米波脉冲频率计接收

```
:CONFigure:CARRier:FREQuency:STATe ON
:CONFigure:CARRier:FREQuency:STATe 1
:CONF:CARR:FREQ:STAT ON
:CONF:CARR:FREQ:STAT 1
```

都是设置手动载波频率功能开。

每一个参数类型都有一个或多个对应的响应数据类型。当查询的时候，数值类型的参数将返回一个实型或者整型响应数据。响应数据是精确的，严格的，被称为“***精确地讲***”。

“***精确地讲***”意味着特定查询的响应格式都是相同的。

例如，如果你查询功率状态（:CONFigure:CARRier:FREQuency:STATe?），当其为开时，不管你先前发送的是 :CONFigure:CARRier:FREQuency:STATe 1 或者:CONFigure:CARRier:FREQuency:STATe ON，响应总是为 1。

表 2-3 参数和响应类型

参数类型	响应数据类型
数值型	实数或者整数
扩展数值型	整数
离散型	离散型
布尔型	数字布尔型
字符串	字符串
非十进制的数值类型	十六进制
	八进制
	二进制

数值参数

分系统命令和普通命令中都可使用数值参数。数值参数接收所有的常用十进制计数法，包括正负号、小数点和科学记数法。如果某一设备只接收指定的数值类型，例如整数，那么它自动将接收的数值参数取整。

以下是数值类型的例子：

100	无小数点
100.	可选小数点
1.23	带符号位
4.56e<space>3	指数标记符 e 后可以带空格
-7.89E-01	指数标记符 e 可以大写或小写
+256	允许前面加正号
.5	小数点可先行

扩展的数值参数

大部分与分系统有关的测量都使用扩展数值参数来指定物理量。扩展数值参数接收所有的数值参数和另外的特殊值。SCPI 命令表中会列出所有有效的参数后缀。注意扩展数值参数不适用于通用命令或者是 STATus 分系统命令。

扩展数值参数举例：

101	数值参数
1.2GHz	GHz 可以被用作指数（E009）
200MHz	MHz 可以被用作指数（E006）
-10ns	-10 纳秒
10%	百分之十

离散型参数

当需要设置的参数值为有限个时，使用离散参数来标识。离散参数使用助记符来表示每一个有效的设置。像程控命令助记符一样，离散参数助记符有长短两种格式，并可使用大小写混合的方式。

下面的例子，离散参数和命令一起使用。

```
:TRIGger:MODE INT|EXT|ARM
```

INT	内部门
EXT	外部门
ARM	ARM 模式

布尔型参数

布尔参数代表一个真或假的二元条件，它只能有四个可能的值。

布尔参数举例：

ON	逻辑真
OFF	逻辑假
1	逻辑真
0	逻辑假

字符串型参数

字符串型参数允许 ASCII 字符串被当成参数发送。单引号和双引号被用作分隔符。

下面是字符串型参数的例子。

```
‘This is Valid’        “This is also Valid”   ‘SO IS THIS’
```

实型响应数据

大部分的测试数据是实数型，其格式可以为基本的十进制计数法或科学计数法，大部分的高级程控语言均支持这两种格式。

实数响应数据举例：

1.23E+0
-1.0E+2
+1.0E+2
0.5E+0
0.23

-100.0
+100.0
0.5

整型响应数据

整数响应数据是包括符号位的整数数值的十进制表达式。当对状态寄存器进行查询时，大多返回整数型响应数据。

整数响应数据事例：

0	符号位可选
+100	允许先行正号
-100	允许先行负号
256	没有小数点

离散响应数据

离散型响应数据和离散型参数基本一样，主要区别是离散型响应数据的返回格式只为大写的短格式。

离散响应数据事例

CW	连续波
PULse	脉冲

数字布尔型响应数据

布尔型的响应数据返回一个二进制的数值 1 或者 0。

字符串型响应数据

字符串响应数据和字符串参数是同样的。主要区别是字符串响应数据的分隔符使用双引号，而不是单引号。字符串响应数据还可嵌入双引号，并且双引号间可以无字符。下面是一些字符串型响应数据的例子

```
“This is a string”
“one double quote inside brackets: (“”)”
```

第七节 SCPI 命令程控消息

下面的命令将用来表达确定的程控消息。

例 1

```
*IDN?
```

SCPI 仪器公用命令,SCPI 把 IEEE488.2 要求仪器必须执行的公用命令作为 SCPI 仪器公用命令，这些公用命令用于控制仪器的某些基本功能操作，其句法和语义遵循 IEEE488.2 的规定

例 2

:CONFigure:MATH:CRF:OFFSet 6GHz 与:CONFigure:MATH:CRF:OFFSet 6 GH

两条指令都是正确的。在命令中可以直接使用测量单位，测量单位与参数可以直接相连也可以加空格。可以接受的单位应在程控手册中做说明。

例 3

```
:CONFigure:CARRier:FREQuency:STATe ON|OFF|0|1
```

布尔数据没有单位，取值 0 或 1

作为字符程控数据时，ON 代表 1，OFF 代表 0

例 4

```
:SYSTem:COMMunicate:LAN:IP <string>
```

字符串行参数，命令可以设置，也可以查询结果。

```
: CALibration:SAVE
```

事件命令，只能设置而不能查询。

第八节 SCPI 命令中数值的进制

命令的值可以用二进制，十进制，十六进制或者八进制的格式输入。当用二进制，十六进制或者八进制时，数值前面需要一个合适的标识符。十进制（默认格式）不需要标识符，当输入一个数值前面没有表示符时，设备会确保其是十进制格式。下面的列表显示了各个格式需要的表示符：

- #B 表示这个数字是一个二进制数值。
- #H 表示这个数字是一个十六进制数值。
- #Q 表示这个数字是一个八进制数值。

下面是 SCPI 命令中十进制数 45 的各种表示：

#B101101

#H2D

#Q55

第三章 SCPI 命令参考

本章包含了所有被本毫米波脉冲频率计识别和执行的 SCPI 命令信息。包括 IEEE488.2 通用命令、标准命令、测量命令集、输入命令集、手动命令集、自动命令集、分辨率命令集、运算命令集和系统命令集等的介绍和说明。

注 1：3212D 毫米波脉冲频率计一次最多可以接收 10 条级联的 SCPI 命令。

注 2：设置通道为通道 1，阻抗为 1MΩ时，只可设置测量模式为连续波，设置测量参数为载波频率和载波周期。设置通道为通道 1，阻抗为 50Ω，设置测量模式为连续波时，可设置测量参数为载波频率和载波周期；设置通道为通道 1，阻抗为 50Ω，设置测量模式为脉冲时，可设置测量参数为载波频率、载波周期、脉冲宽度、脉冲周期、脉冲频率、脉冲关断、脉冲占空比。设置通道为通道 2，只可设置阻抗为 50Ω，设置测量模式为连续波时，可设置测量参数为载波频率和载波周期；设置通道为通道 2，只可设置阻抗为 50Ω，设置测量模式为脉冲时，可设置测量参数为载波频率、载波周期、脉冲宽度、脉冲周期、脉冲频率、脉冲关断、脉冲占空比。

第一节 IEEE488.2 通用命令

通用命令用来控制仪器状态寄存器、状态报告、同步、数据存储及其它通用功能。所有的通用命令都可以通过命令字中的第一个“*”被识别，在 IEEE488.2 中详细定义了这些通用命令。

以下是 IEEE488.2 通用命令的解释和说明。

*CLS

```
*CLS
```

功能描述： 清除状态字节寄存器、数据查询寄存器、标准事件状态寄存器及标准操作状态寄存器。

*ESE

```
*ESE <data>
```

功能描述： 设置标准事件状态使能寄存器。
范围： 0 – 255

*ESE?

```
*ESE?
```

功能描述： 查询标准事件状态使能寄存器的值。

*ESR?

```
*ESR?
```

功能描述： 查询标准事件状态寄存器的值。
范围： 0 – 255

注意 **该命令只能查询。该命令将执行一个破坏性的读，即标准事件状态寄存器的值将被放到一个缓冲区中，直到该数据被查询成功，随即，该寄存器将被清空。**

---

*IDN?

```
*IDN?
```

功能描述： 返回仪器识别字符串，字符串格式如下：
<生产厂商>，<仪器型号>，<串号>，<固件版本号>

注意 **该命令只能查询。**

---

*OPC

*OPC

功能描述： 当所有阻塞操作都完成后，该命令将标准事件状态寄存器的第 0 位设置为‘0’。

*OPC?

*OPC?

功能描述： 查询操作是否完成，返回 1 表明完成，0 表示未完成。

*RST

*RST

功能描述： 将设备的大部分功能被置为厂家预先定义的已知状态。

注意 **该命令只能设置，不能查询。本章的所有命令都给出了其值在复位时的状态。**

*SRE

*SRE <data>

功能描述： 设置服务请求使能寄存器。
范围： 0 - 255

*SRE?

*SRE?

功能描述： 查询服务请求使能寄存器的值。
范围： 0 – 63 或 128 - 191

*STB?

*STB?

功能描述： 查询状态字节寄存器的值。
范围： 0 -255

注意 **该命令只能查询。**

*TST?

*TST?

功能描述： 执行自测试，并且返回自测试结果。

参数说明：

0： 表示所有测试均通过，系统目前工作正常；

1： 表示有至少一项测试未通过，系统目前工作异常。

注意 **该命令只能查询。**

*WAI

*WAI

功能描述： 等待操作完成命令，在仪器执行别的命令操作前会等待前面的操作命令完成。

第二节 标准命令

该子类命令管理设置了状态字节寄存器系统中的错误队列、操作状态寄存器组及数据查询状态寄存器组等。分别如下说明：

:SYSTem:ERRor[:NEXT]?

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

功能描述： 查询错误队列中的最近错误信息，同时删除这个错误信息。若当前没有错误，返回：0,"No error"，若有错误，返回格式：error number,"error description"。

注意 该命令只能查询。

:SYSTem:VERSion?

```
:SYSTem:VERSion?
```

功能描述： 查询仪器遵循的 SCPI 规范的版本号，返回结果如：1999.0。

注意 该命令只能查询。

:STATus:QUEStionable:CONDition?

```
:STATus:QUEStionable:CONDition?
```

功能描述： 该查询返回问题条件数据寄存器的位值的和。例如，若时基未热（bit 4），将返回一个值 16。寄存器中的数据是连续变化的，影响着当前的条件。

范围： 0 – 32767

注意 该命令只能查询。

:STATus:QUEStionable:PTRansition

```
:STATus:QUEStionable:PTRansition <val>
:STATus:QUEStionable:PTRansition?
```

功能描述： 该命令决定当位正向变化时（0 到 1），问题条件数据寄存器中的哪些位将设置问题事件数据寄存中对应的位。参数<val>是你想要设置的位值的和。

范围： 0 – 32767

:STATus:QUEStionable:NTRansition

```
:STATus:QUEStionable:NTRansition <val>
```

```
:STATus:QUEStionable:NTRansition?
```

功能描述： 该命令决定当位负向变化时（1 到 0），问题条件数据寄存器中的哪些位将设置问题事件数据寄存中对应的位。参数<val>是你想要设置的位值的和。

范围： 0 – 32767

:STATus:QUEStionable[:EVENt]?

```
:STATus:QUEStionable[:EVENt]?
```

功能描述： 该查询返回问题事件数据寄存器的位值的和。在条件寄存器能够设置事件寄存器相应位之前，必须设置等价的 PTR 或者 NTR 滤波器。

范围： 0 – 32767

注意 **这是个破坏性的读取命令。在查询前，数据锁存在寄存器中，一旦查询，数据将被清除。**

:STATus:QUEStionable:ENABle

```
:STATus:QUEStionable:ENABle <val>
```

```
:STATus:QUEStionable:ENABle?
```

功能描述： 该命令决定问题事件数据寄存器的哪些位将设置状态字节寄存器的问题状态组的概要位（bit 3）。参数<val>是你想要设置的位值的和。

范围： 0 – 32767

:STATus:OPERation:CONDition?

```
:STATus:OPERation:CONDition?
```

功能描述： 该查询返回操作条件数据寄存器的位值的和。

范围： 0 – 32767

注意 **该命令只能查询。**

:STATus:OPERation:PTRansition

```
:STATus:OPERation:PTRansition <val>
:STATus:OPERation:PTRansition?
```

功能描述： 该命令决定当位正向变化时（0 到 1），标准操作条件寄存器中的哪些位将设置标准操作事件寄存中对应的位。参数<val>是你想要设置的位值的和。

范围： 0 – 32767

:STATus:OPERation:NTRansition

```
:STATus:OPERation:NTRansition <val>
:STATus:OPERation:NTRansition?
```

功能描述： 该命令决定当位负向变化时（1 到 0），标准操作条件寄存器中的哪些位将设置标准操作事件寄存中对应的位。参数<val>是你想要设置的位值的和。

范围： 0 – 32767

:STATus:OPERation[:EVENt]?

```
:STATus:OPERation[:EVENT]?
```

功能描述： 从标准操作事件寄存器返回一个 10 进制的数据

范围： 0 - 32767

注意 **该命令只能查询。该命令将执行一个破坏性的读，该数据放在寄存器中，直到查询该数据为止，随即，该数据将被清空。**

:STATus:OPERation:ENABle

```
:STATus:OPERation:ENABle <val>
:STATus:OPERation:ENABle?
```

功能描述： 该命令决定标准操作事件寄存器中的哪一位将设置在状态字节寄存器中的标准操作状态概要位。

范围： 0 – 32767

:STATus:PRESet

```
:STATus:PRESet
```

功能描述： 该命令复位所有的传输滤波器、使能寄存器和错误/事件队列使能寄存器。

第三节 测量命令集

该子类与测量参数和测量模式有关，包含：设置测量模式、设置脉冲测量模式和设置测量参数。命令集命令和参数如下：

:CONFigure:MEASure:MODE

`:CONFigure:MEASure:MODE CW|PULse`

`:CONFigure:MEASure:MODE?`

功能描述： 设置测量模式：连续波 或 脉冲，查询测量模式。
参数说明： CW 连续波
PULse 脉冲
复位： CW
查询：:CONFigure:MEASure:MODE?，返回值：CW，PUL
按键路径：【测量】[测量模式 连续波 脉冲]

:CONFigure:MEASure:PULSe:MODE

`:CONFigure:MEASure:PULSe:MODE NORM|NPUL`

`:CONFigure:MEASure:PULSe:MODE?`

功能描述：设置脉冲测量模式：正常 或 窄脉冲，查询脉冲测量模式。
参数说明： NORM 正常
NPUL 窄脉冲
复位： NORM
查询：:CONFigure:MEASure:PULSe:MODE?，返回值：NORM，NPUL
按键路径：【测量】 [脉冲测量模式 正常 窄脉冲]

:CONFigure:MEASure:TYPE

`:CONFigure:MEASure:TYPE CRF|CRI|PWID|PRI|PRF|OFFT|DUTY`

功能描述：设置和查询测量参数类型：载波频率、载波周期、脉冲宽度、脉冲周期、脉冲频率、脉冲关断、脉冲占空比。

参数说明： `CRF` 载波频率，`CRI` 载波周期，`PWID` 脉冲宽度，`PRI` 脉冲周期，`PRF`

脉冲频率，`OFFT` 脉冲关断，`DUTY` 脉冲占空比

复位： `CRF`

查询： `:CONFigure:MEASure:TYPE?`，分别对应设置参数。

按键路径：【测量】[测量参数] [载波频率] [载波周期] [脉冲宽度] [脉冲周期] [脉冲频率] [脉冲关断] [脉冲占空比]

:CONFigure:MEASure:REDO

:CONFigure:MEASure:REDO

功能描述：重新测量,事件命令。

参数：无。

:FETCh:CARRier:FREQuency?

:FETCh:CARRier:FREQuency?

功能描述：查询当前载波频率值，单位默认为 Hz。

说明：查询前，先将测量参数类型设置为载波频率，再查询当前载波频率。

:FETCh:CARRier:PERiod?

:FETCh:CARRier:PERiod?

功能描述：查询当前脉冲宽度值，单位默认为 ns。

说明:查询前，先将测量参数类型设置为脉冲宽度，再查询当前脉冲宽度。

:FETCh:PULSe:WIDth?

:FETCh:PULSe:WIDth?

功能描述：查询当前脉冲宽度值，单位默认为 ns。

说明:查询前，先将测量参数类型设置为脉冲宽度，再查询当前脉冲宽度。

:FETCh:PULSe:PERiod?

:FETCh:PULSe:PERiod?

功能描述：查询当前脉冲周期值，单位默认为 ns。

说明:查询前，先将测量参数类型设置为脉冲周期，再查询当前脉冲周期。

:FETCh:PULSe:FREQuency?

:FETCh:PULSe:FREQuency?

功能描述：查询当前脉冲频率值，单位默认为 Hz。

说明:查询前，先将测量参数类型设置为脉冲频率，再查询当前脉冲频率。

:FETCh:PULSe:OFFT?

:FETCh:PULSe:OFFT?

功能描述：查询当前脉冲关断值，单位默认为 ns。

说明：查询前，先将测量参数类型设置为脉冲关断，再查询当前脉冲关断。

:FETCh:PULSe:DUTY?

:FETCh:PULSe:DUTY?

功能描述：查询当前脉冲占空比值，单位默认为%。

说明：查询前，先将测量参数类型设置为脉冲占空比，再查询当前脉冲占空比。

第四节 输入命令集

该子类可设置测量通道和通道阻抗。命令集命令和参数如下：

:CONFigure:FREQuency

```
:CONFigure:FREQuency (@1)|(@2)
```

```
:CONFigure:FREQuency?
```

功能描述：设置和查询测量通道:通道 1 或 通道 2,其中可设置通道 1 的阻抗为 50Ω 或 1MΩ，通道 2。

参数说明：(@1) 通道 1
(@2) 通道 2

复位 (@2)

查询：:CONFigure:FREQuency?，返回值：(@1)，(@2)

按键路径:【输入】 [通道 通道 1 通道 2]

:INPut1:IMPedance

```
:INPut1:IMPedance 50OHM|1MOHM
```

```
:INPut1:IMPedance?
```

功能描述：设置和查询通道 1 阻抗：50Ω 或 1MΩ。

参数说明： 50Ω
1MΩ

复位 50Ω

查询：:INPut1:IMPedance?，返回值：50OHM，1MOHM

按键路径:【输入】 [通道 1 阻抗 50Ω 1MΩ]

第五节　手动命令集

该子类设置自动手动测量方式和自动测量的波段。命令集和参数如下：

:CONFigure:CARRier:FREQuency:STATe

```
:CONFigure:CARRier:FREQuency:STATe ON|OFF|1|0
:CONFigure:CARRier:FREQuency:STATe?
```

功能描述：设置和查询手动载波开关：关 或 开
参数说明：ON|1
　　OFF|0
复位　　OFF
查询：**:CONFigure:CARRier:FREQuency:STATe?**，返回值：0，1
按键路径:【手动】 [载波频率] [手动载波]

:CONFigure:CARRier:FREQuency

```
:CONFigure:CARRier:FREQuency <val><freq unit>
:CONFigure:CARRier:FREQuency?
```

参数：<val><freq unit>
功能描述：设置和查询手动载波频率。
复位：1GHz
范围：　[500MHz，40GHz]
按键路径:【手动】 [载波频率] [载波频率]

:CONFigure:IF:STATe

```
:CONFigure:IF:STATe ON|OFF|1|0
:CONFigure:CARRier:FREQuency:STATe?
```

功能描述：设置和查询手动中频开关：关 或 开
参数说明：　ON|1
　　OFF|0
复位：　　OFF
查询：**:CONFigure:IF:STATe?**，返回值：0，1
按键路径:【手动】 [中频频率] [手动中频]

:CONFigure:IF

```
:CONFigure:IF<val><freq unit>
```

`:CONFigure:IF?`

功能描述：设置和查询手动中频频率
复位：80MHz
范围： [35MHz，120MHz]
按键路径:【手动】 [中频频率] [中频频率]

第六节 自动命令集

该子类设置自动手动测量方式和自动测量的波段。命令集和参数如下：

:CONFigure:AUTO:STAT

:CONFigure:AUTO:STAT ON|OFF|1|0

:CONFigure:AUTO:STAT?

功能描述：设置和查询自动手动测量：自动 或 手动。

参数：ON|OFF|1|0

查询：:CONFigure:AUTO:STAT?，返回值：0，1

:CONFigure:AUTO:BAND

:CONFigure:AUTO:BAND ALL|B0|B1|B2|B3|USER

:CONFigure:AUTO:BAND?

功能描述：设置和查询自动波段：

全波段|0.5—10GHz|10—20GHz|20—26.5GHz|26.5—40GHz|用户定义

参数：

ALL|B0|B1|B2|B3|USER

复位:全波段

查询：:CONFigure:AUTO:BAND?，返回值与设置参数依次对应。

按键路径:【自动】 [波段选择] [全波段] [0.5—10GHz] [10—20GHz] [20—26.5GHz] [26.5—40GHz]||[用户定义]

:CONFigure:USER:FREQuency:STARt

:CONFigure:USER:FREQuency:STARt <val><freq unit>;

:CONFigure:USER:FREQuency:STARt?

功能描述：设置和查询用户定义波段起始频率

复位：8GHz
范围： [500MHz，40GHz]
按键路径:【自动】 [起始频率]

:CONFigure:USER:FREQuency:STOP

:CONFigure:USER:FREQ:STOP <val><freq unit>;

:CONFigure:USER:FREQ:STOP?

功能描述：设置和查询用户定义波段终止频率

复位：10GHz

范围： [500MHz，40GHz]

按键路径:【自动】 [终止频率]

第七节　分辨率命令集

该子类可设置测量结果分辨率，命令集和参数如下：

[:SENSe]:FREQuency:RESolution

[:SENSe]:FREQuency:RESolution 0.1Hz|1Hz|10Hz|100Hz|1kHz|10kHz|100kHz|1MHz

[:SENSe]:FREQuency:RESolution?;

功能描述：设置和查询分辨率。分辨率为 0.1Hz、1Hz、10Hz、100Hz、1kHz、10kHz、100kHz、1MHz。

参数说明：0.1Hz、1Hz、10Hz、100Hz、1kHz、10kHz、100kHz、1MHz

复位:　　1MHz

查询：[:SENSe]:FREQuency:RESolution?，返回值与设置参数依次对应。

按键路径:　【分辨率】[分辨率]

[:SENSe]:AVERage:COUNt

[:SENSe]:AVERage:COUNt <val>

[:SENSe]:AVERage:COUNt?

功能描述：设置和查询平均次数。
复位:　　1
按键路径:【分辨率】 [平均次数]

[:SENSe]:SMOOth:STATe

[:SENSe]:SMOOth:STATe ON|OFF|1|0

[:SENSe]:SMOOth:STATe?

功能描述：设置和查询平滑开关：关 或 开。
复位:　　关

查询：[:SENSe]:SMOOth:STATe?，返回值：0，1

按键路径:【分辨率】 [平滑 关 开]

第八节 运算命令集

该子类可设置 7 个测量参数：“载波频率”、“载波周期”、“脉冲频率”、“脉冲周期”、“脉冲宽度”、“脉冲关断”及“脉冲占空比”的比例和偏置值，频率计根据测量结果，自动计算并显示最终测量结果。命令集和参数如下：

:CONFigure:MATH:CRF:SCALe

:CONFigure:MATH:CRF:SCALe<val>

:CONFigure:MATH:CRF:SCALe?

功能描述：设置和查询载波频率比例。

参数：<val>

复位：1

范围：[-20，20]

按键路径:【运算】[载波频率] [比例]

:CONFigure:MATH:CRF:OFFSet

:CONFigure:MATH:CRF:OFFSet<val>

:CONFigure:MATH:CRF:OFFSet?

功能描述：设置和查询载波频率偏置。

参数：<val><unit>,单位默认为 Hz。

复位：0

范围：[-100GHz, 100GHz]

按键路径:【运算】[载波频率] [偏置]

:CONFigure:MATH:CRI:SCALe

:CONFigure:MATH:CRI:SCALe<val>

:CONFigure:MATH:CRI:SCALe?

功能描述：设置和查询载波周期比例。

参数：<val>

范围：[-20，20]

复位：1

按键路径:【运算】[载波周期] [比例]

:CONFigure:MATH:CRI:OFFSet

:CONFigure:MATH:CRI:OFFSet?

:CONFigure:MATH:CRI:OFFSet<val><unit>

功能描述：设置和查询载波周期偏置,。

参数：<val><unit>,单位默认为 ns

复位: 0

范围：[-100ms, 100ms]

按键路径【运算】[载波周期] [偏置]

:CONFigure:MATH:PRF:SCALe

:CONFigure:MATH:PRF:SCALe<val>

:CONFigure:MATH:PRF:SCALe?

功能描述：设置和查询脉冲频率比例。

参数：<val>

复位: 1

范围：[-20，20]

按键路径 :【运算】[脉冲频率] [比例]

:CONFigure:MATH:PRF:OFFSet

:CONFigure:MATH:PRF:OFFSet<val><unit>

:CONFigure:MATH:PRF:OFFSet?

功能描述：设置和查询脉冲频率偏置。

参数：<val><unit>,单位默认为 Hz

复位: 0

范围：[-100GHz, 100GHz]

按键路径:【运算】[脉冲频率] [偏置]

:CONFigure:MATH:PRI:SCALe

:CONFigure:MATH:PRI:SCALe?

:CONFigure:MATH:PRI:SCALe?

功能描述：设置和查询脉冲周期比例。

参数：<val>

复位：1

范围：[-20，20]

按键路径:【运算】[脉冲周期] [比例]

:CONFigure:MATH:PRI:OFFSet

:CONFigure:MATH:PRI:OFFSet<val><unit>

:CONFigure:MATH:PRI:OFFSet?

功能描述：设置和查询脉冲周期偏置,

参数：<val><unit>，单位默认为 ns

范围：[-100s, 100s]

按键路径:【运算】[脉冲周期] [偏置]

:CONFigure:MATH:PWID:SCALe

:CONFigure:MATH:PWID:SCALe<val>

:CONFigure:MATH:PWID:SCALe?

功能描述：设置和查询脉冲宽度比例。

参数：<val>

复位：1

范围：[-20，20]

按键路径:【运算】[脉冲宽度] [比例]

:CONFigure:MATH:PWID:OFFSet

:CONFigure:MATH:PWID:OFFSet<val><unit>

:CONFigure:MATH:PWID:OFFSet?

功能描述：设置和查询脉冲宽度偏置,

参数：<val><unit>,单位默认为 ns

复位：0

范围：[-100s, 100s]

按键路径:【运算】[脉冲宽度] [偏置]

:CONFigure:MATH:OFFT:SCALe

:CONFigure:MATH:OFFT:SCALe <val>

:CONFigure:MATH:OFFT:SCALe?

功能描述：设置和查询脉冲关断比例。

参数：<val>

复位：1

范围：[-20，20]

按键路径:【运算】[脉冲空时间] [比例]

:CONFigure:MATH:OFFT:OFFSet

:CONFigure:MATH:OFFT:OFFSet<val><unit>

:CONFigure:MATH:OFFT:OFFSet?

功能描述：设置和查询脉冲关断偏置。

参数：<val><unit>

复位：0

范围：[-100s, 100s], 单位默认为 ns

按键路径:【运算】[脉冲空时间] [偏置]

:CONFigure:MATH:DUTY:SCALe

:CONFigure:MATH:DUTY:SCALe<val>

:CONFigure:MATH:DUTY:SCALe?

功能描述：设置和查询脉冲占空比比例。

参数：<val>

复位：1

范围：[-20，20]。

按键路径:【运算】[脉冲占空比] [比例]

:CONFigure:MATH:DUTY:OFFSet

:CONFigure:MATH:DUTY:OFFSet?

:CONFigure:MATH:DUTY:OFFSet<val><unit>

功能描述：设置和查询脉冲占空比偏置。

参数：<val><unit>,单位默认为%

复位：0

范围：[-100%, 100%]

按键路径:【运算】[脉冲占空比] [偏置]

第九节　系统命令集

该子类可设置触发、波形观察、语言、程控、校准等。命令集和参数如下：

:TRIGger:MODE

:TRIGger:MODE INT|EXT|ARM

:TRIGger:MODE?

功能描述： 设置和查询触发方式：内部门模式、外部门模式、ARM 模式。

参数： INT 内部门模式

EXT 外部门模式

ARM。ARM 模式

复位： 内部门模式

查询：:TRIGger:MODE?，返回值，与设置参数依次对应。

按键路径:【系统】[触发] [触发模式] [内部门模式][外部门模式][ARM 模式]

:TRIGger:DELAy

:TRIGger:DELAy <val><unit>

:TRIGger:DELAy?

功能描述：设置和查询触发延时

参数：s|ms|us|ns

参数：<val><unit>,单位默认为 ns

复位：0

范围：[0,1s]

按键路径:【系统】 [触发] [触发延时]

:SYSTem:SCOPe:STATe

:SYSTem:SCOPe:STATe ON|OFF|1|0

:SYSTem:SCOPe:STATe?

功能描述：设置和查询波形观察开关 关 开。

参数：ON|OFF|1|0

查询：:SYSTem:SCOPe:STATe?，返回值：0，1。

:CONFigure:DISPlay:LANGuage

:CONFigure:DISPlay:LANGuage CN|EN

:CONFigure:DISPlay:LANGuage?

功能描述：设置和查询显示语言：中文 或 英文。

参数说明：CN 中文

EN 英文

复位： 中文

查询：返回值：CN（中文），EN（英文）

按键路径:【系统】 [语言 中文 英文]

:SYSTem:COMMunicate:GPIB:ADDRess

:SYSTem:COMMunicate:GPIB:ADDRess <val>

:SYSTem:COMMunicate:GPIB:ADDRess?

功能描述：设置和查询 GPIB 地址。

范围：[0，30]。

复位：1

按键路径:【系统】 [程控][GPIB 地址]

:SYSTem:COMMunicate:LAN:IP

:SYSTem:COMMunicate:LAN:IP <string>

:SYSTem:COMMunicate:LAN:IP?

参数："<string>"

功能描述：设置和查询 IP 地址。

按键路径:【系统】 [程控][网络][IP 地址]

:SYSTem:COMMunicate:LAN: MASK

:SYSTem:COMMunicate:LAN:MASK<string>

:SYSTem:COMMunicate:LAN:MASK?

参数："<string>"

功能描述：设置和查询子网掩码。

按键路径:【系统】 [程控][网络][子网掩码]

:SYSTem:COMMunicate:LAN:GATEway

:SYSTem:COMMunicate:LAN:GATEway <string>

:SYSTem:COMMunicate:LAN:GATEway?

参数："<string>"

功能描述：设置和查询网关。

按键路径:【系统】 [程控][网络][网关]

:SYSTem:COMMunicate:LAN:DNS

:SYSTem:COMMunicate:LAN:DNS <string>

:SYSTem:COMMunicate:LAN:DNS?

参数："<string>"

功能描述：设置和查询 DNS。

按键路径:【系统】 [程控][网络][DNS]

:CALibration:INSertion:STARt

:CALibration:INSertion:STARt

功能描述：启动内插校准。

参数：无。

:CALibration:SAVE

:CALibration:SAVE;

功能描述：存储校准结果。

参数：无。

:SYSTem:PRESet:TYPE

:SYSTem:PRESet:TYPE NORMal USER

:SYSTem:PRESet:TYPE?

功能描述： 复位用户设置：系统 或 用户，查询复位模式。
参数说明： NORMal 系统
USER 用户
复位：NORMal

查询：:SYSTem:PRESet:TYPE?返回值：NORM，USER

按键路径：【系统】[复位设置] [复位模式 厂家 用户]

:SYSTem:PRESet[:USER]:SAVE

:SYSTem:PRESet[:USER]:SAVE

功能描述：存储用户状态。

按键路径：【系统】[复位设置] [存储用户状态]

:SYSTem:SECurity:KB

:SYSTem:SECurity:KB OFF 0

功能描述： 关闭键盘，只可设置不可查询的命令。
参数说明： OFF|0
按键路径：【系统】[安全保密] [关闭键盘]

:SYSTem:SECurity:DISPlay

:SYSTem:SECurity:DISPlay OFF 0

功能描述： 关闭显示器，只可设置不可查询的命令。
参数说明： OFF|0
按键路径：【系统】[安全保密] [关闭显示]

第四章 错误说明

本章主要讲解了本毫米波脉冲频率计远控错误的显示和处理机制、错误的分类方法和原则以及本频率计所能处理的错误的详细信息。

第一节 错误信息概述

当频率计发生错误时，SCPI （远程接口）错误队列和前面板显示错误队列都会报告错误。两个队列在显示和处理方面相互独立，关于前面板错误队列的详细信息请参考用户手册。

注意 **关于连接问题的详细信息，参考相应库文件的帮助文档。**

当使用 SCPI（远程接口）错误队列访问错误消息时，错误队列响应的错误号和<错误描述>显示在主机终端。

注意 **当出现一个前面板上无法显示的错误时，在前面板的状态提示区会显示一个“错误”按钮，以提示用户仪器当前的远控处理检测到错误。**

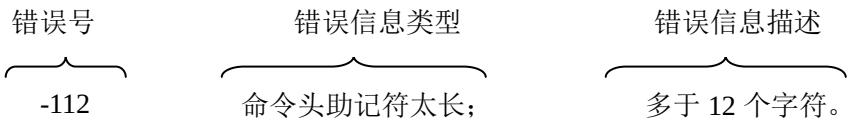
表 4-1 前面板和远控接口错误处理汇总表

项目	前面板错误消息队列	SCPI 远控接口错误消息队列
队列长度（最大保存错误消息数）	100	100
溢出时的处理	循环处理（用新的错误覆盖旧的）	循环处理（用新的错误覆盖旧的）
查看所有错误	通过前面板菜单:【系统】-[信息]，通过上下箭头选择	通过 SCPI 查询命令： SYSTem:ERRor[:NEXT]?
队列清空	通过前面板菜单:【系统】-[信息]-[全部清除]	通过下述三种途径可清空远控接口错误队列中的错误： 仪器重新上电 发送*CLS 命令 查询完最后一条错误
仪器故障信息的处理	错误消息队列清空后重新报告	错误消息队列清空后重新报告

第二节　错误信息格式

前面板显示错误信息时，会显示一个错误号以及该错误号所对应的错误类型信息，同时，还可能会有该错误信息的说明信息（是否显示该信息，视具体错误而定）。

错误信息的格式如下图所示。



另外，本章的第四节中，专门对每一个错误的错误号、错误信息的类型以及该错误的相关信息给出了详细的说明。

第三节 错误信息类型

4.3.1 查询错误

当设备输出队列控制器检测到 IEEE 488.2 描述的信息交换问题。该类错误设置事件状态寄存器（IEEE 488.2，11.5.1 节）的查询错误位（bit 2）。这些错误对应于 IEEE 488.2，6.5 描述的消息交换协议错误。下面的情况下会出现查询错误：

1）、尝试从无数据的输出队列中读取

2）、输出队列数据丢失。

4.3.2 设备相关错误

说明设备操作没有成功完成，可能是因为硬件故障或者固件错误。这些代码同样用于自检响应错误。该类错误设置事件状态寄存器（IEEE 488.2）的设备相关错误位（bit 3）。该类型错误在 SCPI 中没有详细定义，由各仪器厂家根据仪器的硬件、固件及软件的运行情况自行定义。

4.3.3 执行错误

说明设备执行控制模块检测到错误。该类错误设置事件状态寄存器（IEEE 488.2，11.5.1 节）执行错误位（bit 4）。

4.3.4 命令错误

说明命令解释器检测到 IEEE 488.2 语法错误。这类错误将会设置事件状态寄存器(IEEE 488.2, 第 11.5.1 节)的命令错误位（bit 5）。

第四节　错误信息详细描述

异常提示：以下提示，可能因用户设置不当、输入信号超出频率计正常范围或配置参数错误导致。

错误号	错误信息	原因
1	通道 1 幅度过载	输入信号幅度过大
2	通道 1 设置与输入信号不匹配	当面板选择测量连续波，而测量时检测到脉冲状态信号指示有脉冲时，会出现该错误。反之当面板选择测量脉冲波，而测量时没有检测到脉冲状态信号指示有脉冲时，也会出现该错误。
3	通道 1 没检测到脉冲信号	当面板选择测量脉冲波，而测量时没有检测到脉冲状态信号指示有脉冲时，会出现该错误。
4	通道 1 测量失败	测量过程没有正常结束。
5	通道 1 脉冲过窄	通道 1 的最小脉冲宽度为 1000ns,当输入信号脉冲宽度小于这一值时，会出现错误信息提示。
6	通道 1 信号不满足测量条件	当脉冲测量时，脉冲包络内脉冲数过少。
7	通道 2 幅度过载	输入信号幅度过大
8	通道 2 设置与输入信号不匹配	当面板选择测量连续波，而测量时检测到脉冲状态信号指示有脉冲时，会出现该错误。反之当面板选择测量脉冲波，而测量时没有检测到脉冲状态信号指示有脉冲时，也会出现该错误。
9	通道 2 没检测到脉冲信号	当面板选择测量脉冲波，而测量时没有检测到脉冲状态信号指示有脉冲时，会出现该错误。
11	通道 2 测量失败	测量过程没有正常结束。
13	通道 2 脉冲过窄，请使用窄脉冲模式	当脉冲宽度小于 200ns，而脉冲测量模式为正常时出现。
14	通道 2 信号不满足测量条件	当脉冲测量时，脉冲包络内脉冲数过少。
15	测量超时	在规定的时限内没有完成测量。
20	参数文件大小错误	参数文件损坏
60	菜单类型错误	菜单配置参数错误
61	菜单硬键索引错误	菜单配置参数错误
70	输入数据类型错误	配置参数错误
71	输入精度参数错误	配置参数错误
100	打开 USB 设备失败	配置参数错误
101	关闭 USB 设备失败	配置参数错误
102	USB 接收数据失败	配置参数错误

103	USB 发送数据失败	配置参数错误
110	打开 TCP 失败	配置参数错误
111	关闭 TCP 失败	配置参数错误
112	TCP 接收数据失败	配置参数错误
113	TCP 发送数据失败	配置参数错误
114	打开网卡设备错误	配置参数错误
115	获得网卡名错误	配置参数错误
116	打开注册表错误	配置参数错误
117	重启网卡错误	配置参数错误
120	门控板 PLL 失锁	内外时基切换时，可能出现此提示。 外时基不够稳定，可能出现此提示，请使用内时基，看此提示是否继续出现。
121	门控板外闸门开启错误	此提示仅在脉冲载波外闸门测量时出现，指外闸门早于调制脉冲开启。请用户调整外闸门或与频率计“外闸门/触发延时”处进行设置，使外闸门完全落于脉冲开期间内。
122	门控板 Arm 开启错误	此提示仅在脉冲载波触发测量时出现，指触发信号在脉冲开期间到来。请用户调整触发信号或与频率计“外闸门/触发延时”处进行设置，使触发信号在脉冲关期间到来。
123	门控板闸门关闭错误	此提示仅在脉冲载波触发测量时出现。 外闸门时，指外闸门晚于调制脉冲关闭。请用户调整外闸门或与频率计“外闸门/触发延时”处进行设置，使外闸门完全落于脉冲开期间内。 内闸门时，指闸门关闭过晚或根本没有关闭，可能诱因有频率切换等。属正常提示，可剔除不可靠测量结果。
136	门控板写校验错	内外时基切换时，可能出现此提示。
138	门控板读校验错	内外时基切换时，可能出现此提示。
-100	命令错	一般为命令串拼写错，或输入的命令不支持。
-100	命令错：命令串空	命令串空
-100	命令错：命令映射表错	参数转换出错
-100	命令错：级联命令个数超过最大允许数	级联命令个数超过最大允许数
-220	参数错	一般为参数拼写错
-220	参数错：不支持该参数	一般为输入了不支持的离散参数。
-220	参数错：参数个数错	参数个数不满足要求。
-220	参数错：参数类型错	参数类型不满足要求。

故障提示：如出以下提示，可能存在硬件故障

错误号	错误信息	原因
124	门控板 Measure_En 关闭错误	此项信号仅在脉冲载波测量时出现，且常与“门控板闸门关闭错误”同时出现，因脉冲包络内没有中频产生。 偶然出现，可能诱因有频率切换等，属正常提示，可剔除不可靠测量结果。 频繁出现，则可能属信号发生器脉冲调制性能异常、调制脉冲过窄或频率计本身故障。用户可尝试以下检验：确认信号发生器脉冲调制功能正常的前提下，设置信号发生器输出载波 3GHz 功率 -5dBm，调制脉冲 10μs/20μs，频率计工作在内闸门脉冲载波测量状态，分辨率 100kHz，看是否此错误提示仍频繁出现。 如上述测试时提示仍频繁出现，请联系厂家返修。
137	门控板纠错失败(100 次)	如果此时频率计工作于外时基模式，请拔掉外时基，看是否仍有此项错误。 如果是内时基状态第一次看到此项错误，请关电等待一段时间后重新启动频率计。如仍有此项错误出现，请联系厂家返修。

第五章 IVI-C 仪器驱动库

仪器驱动库函数是对 SCPI 命令及底层操作进行了封装，用户可以不必关心底层使用的通信协议就可以实现对 3212D 脉冲频率计进行控制。方便了用户使用，仪器驱动库提供的结果形式包括：驱动库动态链接库文件（*.dll）、导入库文件（*.lib）、驱动库头文件（*.h）、函数面板文件（*.fp）、属性信息文件（*.sub）。

为方便用户使用 3212D 脉冲频率计 IVI-C 仪器驱动库，下面对驱动库函数和属性进行详细介绍：

- ◆ 驱动库函数
- ◆ 驱动库属性

注：设置通道为通道 1，阻抗为 1MΩ时，只可设置测量模式为连续波，设置测量参数为载波频率和载波周期。设置通道为通道 1，阻抗为 50Ω，设置测量模式为连续波时，可设置测量参数为载波频率和载波周期；设置通道为通道 1，阻抗为 50Ω，设置测量模式为脉冲时，可设置测量参数为载波频率、载波周期、脉冲宽度、脉冲周期、脉冲频率、脉冲关断、脉冲占空比。设置通道为通道 2，只可设置阻抗为 50Ω，设置测量模式为连续波时，可设置测量参数为载波频率和载波周期；设置通道为通道 2，只可设置阻抗为 50Ω，设置测量模式为脉冲时，可设置测量参数为载波频率、载波周期、脉冲宽度、脉冲周期、脉冲频率、脉冲关断、脉冲占空比。

第一节 驱动库安装和使用

5.1.1 使用示例之前

为了能正确使用以下示例，在安装 IVI 驱动程序（例如，IVI-COM 和 IVI-C）之前，必须先安装 IVI Shared Components 和 NI VISA 或 Agilent VISA 库。可以通过安装 Agilent IO Libraries 进行上述功能的安装。

5.1.2 安装 3212D 仪器驱动库

打开 3212D 脉冲频率计 IVI 驱动安装文件，点击“Setup.exe”，按照安装向导提示步骤安装即可。下表详细列出了 3212D 仪器驱动库的安装路径及文件。其中<IviInstallDir>为 IVI Shared Components 安装路径，默认为“C:\Program Files\IVI Foundation\IVI”。仪器驱动库默认安装路径如下表：

文件	描述
<IviInstallDir>\Bin\3212D.dll	动态链接库文件
<IviInstallDir>\Include\3212D.h	驱动头文件
<IviInstallDir>\Lib\msc\3212D.lib	微软格式导入库文件
<IviInstallDir>\Drivers\3212D\3212D.fp	函数面板文件
<IviInstallDir>\Drivers\3212D\3212D.sub	属性信息文件
<IviInstallDir>\Drivers\3212D\3212D 脉冲频率计程控手册.pdf	帮助文件

5.1.3 在 LabWindows CVI 9.0 中的使用方法

在使用 IVI-C 仪器驱动库时，必须包含“3212D.h”头文件和正确导入“3212D.lib”库文件。下面介绍该驱动库在 LabWindows CVI 9.0 中的使用步骤：

（1） 从“Edit”菜单选择“Add File to Project…”，然后选择“3212D.fp”；

（2） 将头文件“3212D.h”、库文件“3212D.lib”和面板文件“3212D.fp”文件导入到测试项目中；

（3） 在测试项目中添加使用的头文件，如“#include “3212D.h”” 文件；

（4） 按上述步骤就可以使用驱动库了。

5.1.4 在 Visual C++ 2005 中的使用方法

在使用 IVI-C 仪器驱动库时，必须包含“3212D.h”头文件和正确导入“3212D.lib”库文件。下面介绍该驱动库在 Visual C++2005 中的使用步骤：

（1） 选择“项目”菜单选择“属性”菜单，然后选择“C/C++”-〉“常规”-〉“附加包含目录”选项，添加编译所需要的头文件所在路径；

（2） 选择“项目”菜单选择“属性”菜单，然后选择“连接器”-〉“常规”-〉“附加库目录”选项，添加链接所需要的库文件所在路径；

（3） 选择“项目”菜单选择“属性”菜单，然后选择“连接器”-〉“输入”-〉“附加依赖项”选项，输入“3212D.lib”添加项目运行时的库文件，也可以通过在测试项目顶部以“#pragma comment(lib,"3212D")”方式添加项目运行时的库文件；

（4） 在测试项目中添加使用的头文件，如“#include “3212D.h”” 文件；

（5） 按上述步骤就可以使用驱动库了。

第二节 驱动库函数

5.2.1 仪器初始化及销毁函数

ViStatus AV3212D_init

ViStatus AV3212D_init (ViRsrc resourceName,ViBoolean IDQuery,ViBoolean resetDevice,
ViSession *vi)

功能描述：该函数执行仪器初始化操作：

-创建一个新的 IVI 仪器驱动 session。

-使用 resourceName 参数中指定的接口和地址打开特定仪器的 session。

-如果 IDQuery 参数为真，该函数查询仪器 ID 并检测其有效性。

-如果 resetDevice 参数设置为真，该函数将仪器复位到一个已知状态。

-发送初始化命令设置仪器操作的必要状态。

-返回 ViSession 句柄用于所有后续函数调用的仪器识别。

参数说明：ViRsrc resourceName：仪器初始化使用的资源字符串，该字符串可以是 IVI 配置应用中指定的逻辑名，也可以是有效的 VISA 地址。假如用户在 NI-MAX 工具中配置的逻辑名若为”myLogicalName”，仪器 GPIB 地址为 15，IP 地址为”172.141.1.1” ，主机名为”3212D”，那么，不同场合下的初始化实例如下：

逻辑名：resourceName = ” myLogicalName”

网络：resourceName = ”TCPIP0::172.141.1.1::5000::SOCKET” 或

resourceName = ”TCPIP0::3212D::5000::SOCKET”

GPIB：resourceName = ”GPIB0::18::INSTR”

ViBoolean IDQuery：指定用户是否需要执行 ID 查询。有效值范围如下：

VI_TRUE (1) - 执行 ID 查询

VI_FALSE (0) - 跳过 ID 查询

ViBoolean resetDevice：指定在初始化过程中是否需要复位仪器。有效取值范围：

VI_TRUE (1) - 复位仪器

VI_FALSE (0) - 不复位仪器

ViSession *vi：返回用于识别仪器的 ViSession 句柄，该仪器的后续函数将使用该句柄调用。

返 回 值：成功或失败错误代码。

ViStatus AV3212D_InitWithOptions

ViStatus AV3212D_InitWithOptions (ViRsrc resourceName,ViBoolean IDQuery,ViBoolean resetDevice, ViConstString optionString, ViSession *vi)

功能描述：该函数执行仪器初始化操作：

-创建一个新的 IVI 仪器驱动 session。

-使用 resourceName 参数中指定的接口和地址打开特定仪器的 session。

-如果 IDQuery 参数为真，该函数查询仪器 ID 并检测其有效性。

-如果 resetDevice 参数设置为真，该函数将仪器复位到一个已知状态。

-发送初始化命令设置仪器操作的必要状态。

-返回 ViSession 句柄用于所有后续函数调用的仪器识别。

参数说明：**ViRsrc resourceName**：仪器初始化使用的资源字符串，该字符串可以是 IVI 配置应用中指定的逻辑名，也可以是物理资源描述符。假如用户在 NI-MAX 工具中配置的逻辑名若为”myLogicalName”，仪器 GPIB 地址为 15，IP 地址为”127.0.0.1”，主机名为”3212D”，那么，不同场合下的初始化实例如下：

逻辑名：resourceName = “myLogicalName”

TCPIP：resourceName = ”TCPIP0:: 127.0.0.1::5000::SOCKET” 或

resourceName = ”TCPIP0::3212D::5000::SOCKET”

GPIB：resourceName = ”GPIB0::18::INSTR”

ViBoolean IDQuery：指定用户是否需要执行 ID 查询。有效值范围如下：

VI_TRUE (1) - 执行 ID 查询

VI_FALSE (0) - 跳过 ID 查询

ViBoolean resetDevice：指定在初始化过程中是否需要复位仪器。有效取值范围：

VI_TRUE (1) - 复位仪器

VI_FALSE (0) - 不复位仪器

ViConstString optionString：用户可以使用该控制设置 session 属性的特定初始化值。字符串的格式为："AttributeName=Value"，其中 AttributeName 是属性的名称，Value 是属性将要设置的值。若要设置多个属性，可以将它们的值多个逗号分隔。如果传递 VI_NULL 或空字符串，该 session 将使用属性的缺省值。用户可以通过明确指定参数值的方式覆盖缺省值，用户不需要指定所有属性，可以忽略任何一个。如果用户未指定某一属性，将会使用其缺省值。下面列出了参数中可识别的属性和名称。

名称	属性定义常量
RangeCheck	AV3212D_ATTR_RANGE_CHECK
QueryInstrStatus	AV3212D_ATTR_QUERY_INSTRUMENT_STATUS
Cache	AV3212D_ATTR_CACHE
Simulate	AV3212D_ATTR_SIMULATE
RecordCoercions	AV3212D_ATTR_RECORD_COERCIONS

ViSession *vi：返回用于识别仪器的 ViSession 句柄，该仪器的后续函数将使用该句柄调用。

返 回 值：成功或失败错误代码。

ViStatus AV3212D_close

ViStatus AV3212D_close (ViSession vi)

功能描述：该函数执行下列操作：

- 关闭仪器 I/O session。

- 销毁仪器驱动 session 及其所有属性。
- 释放该驱动使用的所有内存资源。

参数说明：ViSession vi，通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。

5.2.2 状态函数

ViStatus AV3212D_StatusClear

ViStatus AV3212D_StatusClear (ViSession vi)

功能描述：清除所有状态寄存器及错误队列。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调用 3212D_ClearError 函数。

程控命令：*CLS

ViStatus AV3212D_StatusPreset

ViStatus AV3212D_StatusPreset (ViSession vi)

功能描述：将状态寄存器恢复到系统默认值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调用 3212D_ClearError 函数。

程控命令：:STATus:PRESet

5.2.3 应用函数

ViStatus AV3212D_reset

ViStatus AV3212D_reset (ViSession vi)

功能描述：将系统恢复到一个默认状态。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调用 3212D_ClearError 函数。

程控命令：*RST

ViStatus AV3212D_ResetWithDefaults

ViStatus AV3212D_ResetWithDefaults (ViSession vi)

功能描述：复位仪器，应用配置存储服务器中的缺省设置。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调用 3212D_ClearError 函数。

程控命令：*RST

ViStatus AV3212D_revision_query

ViStatus AV3212D_revision_query (ViSession vi, ViChar instrumentDriverRevision[],

ViChar firmwareRevision[])

功能描述：返回驱动和仪器版本。

参数说明：

ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViChar instrumentDriverRevision[]：存储待返回的驱动版本。

ViChar firmwareRevision[]：存储待返回的仪器版本。

返 回 值：

成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调用 3212D_ClearError 函数。

程控命令：

*IDN?

ViStatus AV3212D_error_query

ViStatus AV3212D_error_query (ViSession vi, ViInt32 *errCode, ViChar errMessage[])

功能描述：查询错误队列中最近的错误信息，同时删除这个错误信息。若当前没有错误，返回：0,"No error"，若有错误，返回格式：error number,"error description"。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViInt32 *errCode：待返回的仪器错误代码。

ViChar errMessage[]：待返回的仪器错误消息。

返 回 值：成功或失败错误代码。可通过 3212D_error_messag 函数获得。为获得有关错误状态的附加错误信息，调用 3212D_GetError 函数。若要清除来自驱动的错误信息，调 3212D_ClearError 函数。

程控命令：:SYSTem:ERRor[:NEXT]?

ViStatus AV3212D_error_message

ViStatus AV3212D_error_message (ViSession vi, ViStatus errorCode,ViChar errorMessage[])

功能描述：将 IVI 驱动返回的错误代码转换成用户可识别的形式。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViStatus errCode：待查询的仪器错误代码。

ViChar errMessage[]：待返回的仪器错误消息。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_InvalidateAllAttributes

ViStatus AV3212D_InvalidateAllAttributes (ViSession vi)

功能描述：该函数验证所有属性的缓冲值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_GetError

ViStatus AV3212D_GetError (ViSession vi, ViStatus *errorCode, ViInt32 bufferSize,ViChar description[])

功能描述：该函数查询并且清除 session 或当前执行线程的错误信息。如果用户通过 vi 参数指定一个有效的 IVI session，那么该函数查询并清除该 session 的错误信息。如果用户为 vi 参数传递一个 VI_NULL，那么该函数查询并清除当前执行线程的错误信息。如果用户传递的 vi 参数无效，该函数什么也不做并返回一个错误。通常，错误信息表述了最新发生的错误。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViStatus *errorCode：待返回的错误代码，”0”,表示没有错误，正值表示警告，负值表示错误。如果用户不想返回该值可以指定 VI_NULL。

ViInt32 bufferSize：错误缓冲区大小。

ViChar description[]：存储返回的错误字符串。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_ClearError

ViStatus AV3212D_ClearError (ViSession vi)

功能描述：该函数清除清除当前执行线程或 IVI session 的错误代码和错误描述。如果用户通过 vi 参数指定一个有效的 IVI session，那么该函数清除该 session 的错误信息。如果用户为 vi 参数传递一个

VI_NULL，那么该函数清除当前执行线程的错误信息。如果用户传递的 vi 参数无效，该函数什么也不做并返回一个错误。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_LockSession

ViStatus AV3212D_LockSession (ViSession vi, ViBoolean *callerHasLock)

功能描述：申请获取驱动 session 的多线程锁，当其它执行线程都释放了仪器 session 的锁之后该函数才能获得该多线程锁。当你使用该函数成功获得多线程锁之后，只有当调用 3212D_UnlockSession 释放该多线程锁后，仪器 session 其它线程才能访问仪器 session。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViBoolean *callerHasLock：该参数用于跟踪复杂函数调用过程，用户可以根据该函数参数判断是否已加锁或解锁。如果该参数返回值为 VI_TRUE，用户调用该函数就不会对驱动 session 加锁，如果该参数返回值为 VI_FALSE，该函数获取该多线程锁，然后设置该参数值为 VI_TRUE。该参数仅是为方便用户便利使用，如果用户不想使用该参数，可以为该参数传递 VI_NULL。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_UnlockSession

ViStatus AV3212D_UnlockSession (ViSession vi, ViBoolean *callerHasLock)

功能描述：该函数释放 3212D_LockSession 函数获取的获取驱动 session 多线程锁。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViBoolean *callerHasLock：该参数用于跟踪复杂函数调用过程，用户可以根据该函数参数判断是否已加锁或解锁。如果该参数返回值为 VI_TRUE，用户调用该函数就不会对驱动 session 加锁，如果该参数返回值为 VI_FALSE，该函数获取该多线程锁，然后设置该参数值为 VI_TRUE。该参数仅是为方便用户便利使用，如果用户不想使用该参数，可以为该参数传递 VI_NULL。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_WriteInstrData

ViStatus AV3212D_WriteInstrData (ViSession vi, ViConstString writeBuffer)

功能描述：向仪器发送指定的命令字符串，该函数绕过 IVI 属性状态缓冲，因此调用该函数时所有属性的缓冲值是无效的。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString writeBuffer：待写入的命令字符串。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_ReadInstrData

ViStatus AV3212D_ReadInstrData (ViSession vi, ViInt32 numBytes, ViChar rdBuf[], ViInt32 *bytesRead)

功能描述：从仪器读数据，该函数绕过 IVI 属性状态缓冲，因此调用该函数时所有属性的缓冲值是无效的。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViInt32 numBytes：读缓冲器大小。

ViChar rdBuf[]：仪器读缓冲器。

ViInt32 *bytesRead：实际读取的字节数。

返 回 值：成功或失败错误代码。

程控命令：无

5.2.4 属性访问函数

ViStatus AV3212D_SetAttributeViInt32

ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

功能描述：该函数用于设置 ViInt32 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“ (@2)”。

ViAttr attributeId：待设置的属性 ID。

ViInt32 value：待设置的属性值。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_SetAttributeViReal64

ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

功能描述：该函数用于设置 ViReal64 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：待设置的属性 ID。

ViReal64 value：待设置的 ViReal64 类型属性值。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_SetAttributeViString

ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViConstString value)

功能描述：该函数用于设置 ViString 属性值。

参数说明：

ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：待设置的属性 ID。

ViReal64 value：待设置的 ViString 类型属性值，若待设置的属性值为空，可以传递 VI_NULL 或空字符串。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_SetAttributeViBoolean

ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

功能描述：该函数用于设置 ViBoolean 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：待设置的属性 ID。

ViBoolean value：待设置的 ViBoolean 类型属性值，取值为 VI_TRUE,VI_FALSE。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_GetAttributeViInt32

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

功能描述：该函数用于查询 ViInt32 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：属性 ID。

ViInt32 *value：待设返回的属性值。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_GetAttributeViReal64

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

功能描述：该函数用于查询 ViReal64 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：属性 ID。

ViReal64 *value：待返回的 ViReal64 类型属性值。

返 回 值：成功或失败错误代码。

程控命令：无

ViStatus AV3212D_GetAttributeViString

ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

功能描述： 该函数用于查询 ViString 属性值。

参数说明： ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId： 待设置的属性 ID。

ViInt32 bufSize： 存储属性值的缓冲区大小。

ViChar value[]： 存储属性值的缓冲区。

返 回 值： 成功或失败错误代码。

程控命令：无

ViStatus AV3212D_GetAttributeViBoolean

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

功能描述：该函数用于查询 ViBoolean 属性值。

参数说明：ViSession vi：通过 3212D_init 或 3212D_InitWithOptions 函数获得的 ViSession 句柄，句柄标识仪器特定的 session。

ViConstString channelName：如果该属性是基于通道的，那么该参数指定属性所在通道，如果属性不是基于通道的，那么可以为该参数指定为 VI_NULL 或空字符串。有效通道取值为通道 1“(@1)”，通道 2“(@2)”。

ViAttr attributeId：属性 ID。

ViBoolean *value：待返回的 ViBoolean 类型属性值。

返 回 值：成功或失败错误代码。

程控命令：无

第三节 驱动库属性

5.3.1 测量

AV3212D_ATTR_CONFIGURE_MEASURE_MODE

描　　述：设置和查询测量模式。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value： 3212D_VAL_CONFIGURE_MEASURE_MODE_CW，连续波
3212D_VAL_CONFIGURE_MEASURE_MODE_PULSE，脉冲

ViInt32 *value: 0，连续波
1，脉冲

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MEASure:MODE { CW | PULse }
:CONFigure:MEASure:MODE?

AV3212D_ATTR_CONFIGURE_MEASURE_PULSE_MODE

描　　述：设置和查询脉冲测量模式。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value:
3212D_VAL_CONFIGURE_MEASURE_PULSE_MODE_NORM，正常
3212D_VAL_CONFIGURE_MEASURE_PULSE_MODE_NPUL，窄脉冲

ViInt32 *value: 0，正常
1，窄脉冲

ViConstString channelName：VI_NULL 或””

程控命令： :CONFigure:MEASure:PULSe:MODE {NORM|NPUL}
:CONFigure:MEASure:PULSe:MODE?

AV3212D_ATTR_CONFIGURE_MEASURE_TYPE

描　　述：设置和查询测量参数类型。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：3212D_VAL_CONFIGURE_MEASURE_TYPE_CRF，载波频率

3212D_VAL_CONFIGURE_MEASURE_TYPE_CRI，载波周期

3212D_VAL_CONFIGURE_MEASURE_TYPE_PWID，脉冲宽度

3212D_VAL_CONFIGURE_MEASURE_TYPE_PRI，脉冲宽度

3212D_VAL_CONFIGURE_MEASURE_TYPE_PRF，脉冲周期

3212D_VAL_CONFIGURE_MEASURE_TYPE_OFFT，脉冲关断

3212D_VAL_CONFIGURE_MEASURE_TYPE_DUTY，脉冲占空比

ViInt32 *value：设置的测量参数类型索引号，0——载波频率、1——载波周期、2——脉冲宽度、3——脉冲周期、4——脉冲频率、5——脉冲关断、6——脉冲占空比。

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:MEASure:TYPE {CRF|CRI PWID|PRI|PRF|OFFT|DUTY}

:CONFigure:MEASure:TYPE?

AV3212D_ATTR_CONFIGURE_MEASURE_REDO

描　　述：启动重新测量。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViConstString value)

参数说明：　ViConstString value：NULL

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:MEASure:REDO

AV3212D_ATTR_ FETCH_CARRIER_FREQUENCY

描　　述：查询当前载波频率值。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value，存储查询返回的载波频率值，单位默认为 Hz。

ViConstString channelName：VI_NULL 或""

程控命令：:FETCh:CARRier:FREQuency?

AV3212D_ATTR_ FETCH_CARRIER_PERIOD

描　　述：查询当前载波周期值。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value 存储查询返回的载波周期值，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：:FETCh:CARRier:PERiod?

AV3212D_ATTR_ FETCH_PULSE_WIDTH

描　　述：查询当前脉冲宽度值。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value，存储查询返回的脉冲宽度值，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：FETCh:PULSe:WIDth?

AV3212D_ATTR_ FETCH_ PULSE_PERIOD

描　　述：查询当前脉冲周期值。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：存储查询返回的脉冲周期值，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：:FETCh:PULSe:PERiod?

AV3212D_ATTR_ FETCH_PULSE_FREQUENCY

描　　述：查询当前脉冲频率值。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value，存储查询返回的脉冲频率值，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：:FETCh:PULSe:FREQuency?

AV3212D_ATTR_ FETCH_PULSE_OFFT

描　　述：查询当前脉冲空时间。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value，查询当前脉冲空时间。

ViConstString channelName：VI_NULL 或””

程控命令：:FETCh:PULSe:OFFT?

AV3212D_ATTR_ FETCH_PULSE_DUTY

描　　述：查询当前脉冲占空比。

访问函数：ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 *value，存储查询返回的脉冲占空比。

ViConstString channelName：VI_NULL 或""

程控命令：:FETCh:PULSe:DUTY?

5.3.2 输入

AV3212D_ATTR_ CONFIGURE_FREQUENCY

描　　述：设置和查询通道。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value:　3212D_VAL_CONFIGURE_FREQUENCY_CH1，通道 1

3212D_VAL_CONFIGURE_FREQUENCY_CH2，通道 2

设置的通道索引号，0——通道 1、1——通道 2。

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:FREQuency { (@1) (@2) }

:CONFigure:FREQuency?

AV3212D_ATTR_ INPUT1_IMPEDANCE

描　　述：设置和查询通道 1 阻抗。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：3212D_VAL_INPUT1_IMPEDANCE_50OHM，50Ω

3212D_VAL_INPUT1_IMPEDANCE_1MOHM，1MΩ

设置的通道阻抗索引号，0——50Ω、1——1MΩ。

ViConstString channelName：VI_NULL 或""

程控命令：:INPut1:IMPedance { 50OHM|1MOHM }

:INPut1:IMPedance?

5.3.3 手动

AV3212D_ATTR_ CONFIGURE_CARRIER_FREQUENCY_STATE

描　　述：设置和查询手动载波频率开关。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

参数说明： ViBoolean value：VI_FALSE，关

VI_TRUE，开

设置的手动载波频率开关状态索引号，0——关、1——开。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:CARRier:FREQuency:STATe { OFF|ON|0|1}

:CONFigure:CARRier:FREQuency:STATe?

AV3212D_ATTR_ CONFIGURE_CARRIER_FREQUENCY

描　　述：设置和查询手动载波频率

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的手动载波频率值，范围【5.0e8，4.0e10】，单位默认为 Hz

ViReal64 *value：返回的触发频率值

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:CARRier:FREQuency {<freq>}

:CONFigure:CARRier:FREQuency?

AV3212D_ATTR_ CONFIGURE_IF_STATE

描　　述：设置和查询手动中频开关。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

参数说明： ViBoolean value：VI_FALSE，关

VI_TRUE，开

设置的手动中频开关状态索引号，0——关、1——开。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:IF:STATe { OFF|ON|0|1}

:CONFigure:IF:STATe?

AV3212D_ATTR_CONFIGURE_IF

描　　述：设置和查询手动中频频率

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：置的手动中频频率值，范围【3.5e7，1.2e8】，单位默认为 Hz

ViReal64 *value：返回的中频频率值

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:IF {<freq>}

:CONFigure:IF?

5.3.4 自动

AV3212D_ATTR_ CONFIGURE_AUTO_STAT

描　　述：设置和查询自动开关。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

参数说明：　ViBoolean value：VI_FALSE，关

VI_TRUE，开

设置的自动开关状态索引号，0——关、1——开。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:AUTO:STATe { OFF|ON|0|1}

:CONFigure:AUTO:STATe?

AV3212D_ATTR_ CONFIGURE_AUTO_BAND

描　　述：设置和查询自动波段。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：3212D_VAL_CONGIGURE_AUTO_BAND_ALL，全波段

3212D_VAL_CONGIGURE_AUTO_BAND_B0，0.5--10GHz

3212D_VAL_CONGIGURE_AUTO_BAND_B1，10--20GHz

3212D_VAL_CONGIGURE_AUTO_BAND_B2，20—26.5GHz

3212D_VAL_CONGIGURE_AUTO_BAND_B2，26.5—40GHz

3212D_VAL_CONGIGURE_AUTO_BAND_USER，用户定义

ViConstString channelName：VI_NULL 或””

返回值描述:

ViInt32 *value：

0，全波段

1，0.5--10GHz

2，10--20GHz

3，20—26.5GHz

4，26.5—40GHz

5，用户定义

程控命令：:CONFigure:AUTO:BAND ALL B0 B1 B2 B3 USER

:CONFigure:AUTO:BAND?

AV3212D_ATTR_ CONFIGURE_USER_FREQUENCY_START

描　　述：设置和查询自定义波段起始频率

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的自定义波段起始频率值，范围【5.0e8，4.0e10】，单位默认为 Hz。

ViReal64 *value：返回的波段起始频率值

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:USER:FREQuency:STARt <val><freq unit>;

:CONFigure:USER:FREQuency:STARt?

AV3212D_ATTR_CONFIGURE_USER_FREQUENCY_STOP

描　　述：设置和查询自定义波段终止频率

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的自定义波段终止频率值，范围【5.0e8，4.0e10】，单位默认为 Hz。

ViReal64 *value：返回的波段起始频率值

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:USER:FREQuency:STOP <val><freq unit>;

:CONFigure:USER:FREQuency:STOP?

5.3.5 分辨率

AV3212D_ATTR_ CONFIGURE_AUTO_BAND

描　　述：设置和查询频率分辨率。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：

3212D_VAL_FREQ_RES_0DOT1HZ，0.1Hz

3212D_VAL_FREQ_RES_1HZ，1Hz

3212D_VAL_FREQ_RES_10HZ，10Hz

3212D_VAL_FREQ_RES_100HZ，100Hz

3212D_VAL_FREQ_RES_1KHZ，1kHz

3212D_VAL_FREQ_RES_10KHZ，10kHz

3212D_VAL_FREQ_RES_100KHZ，100kHz

3212D_VAL_FREQ_RES_1MHZ，1MHz

`　　设置的频率分辨率索引号，0——0.1Hz、1——1Hz、2——10Hz、3——100Hz、4——1kHz、5——10kHz、6——100kHz、7——1MHz。

ViConstString channelName：VI_NULL 或””

程控命令：[:SENSe]:FREQuency:RESolution{0.1Hz|1Hz|10Hz|100Hz|1kHz|10kHz|100kHz|1MHz}

[:SENSe]:FREQuency:RESolution?

AV3212D_ATTR_SENSE_AVERAGE_COUNT

描　　述：设置和查询平均次数。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：设置的平均次数，范围【1.1000】。

ViConstString channelName：VI_NULL 或””

程控命令：[:SENSe]:AVERage:COUNt val

[:SENSe]:AVERage:COUNt?

3212D_ ATTR_SENSE_SMOOTH_STATE

描　　述：设置和查询自动开关。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

参数说明： ViBoolean value：VI_FALSE，关

VI_TRUE，开

设置的平滑开关状态索引号，0——关、1——开。

ViConstString channelName：VI_NULL 或””

程控命令：[:SENSe]:SMOOth:STATe　{ OFF|ON|0|1}

[:SENSe]:SMOOth:STATe?

5.3.6 运算

AV3212D_ATTR_ CONFIGURE_MATH_CRF_SCALE

描　　述：设置和查询载波频率比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的载波频率比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:CRF:SCALe <val>

:CONFigure:MATH:CRF:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_CRF_OFFSET

描　　述：设置和查询载波周期偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设设置的载波频率偏置，范围【-1.0e11，1.0e11】，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:CRF:OFFSet <val>

:CONFigure:MATH:CRF:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_CRI_SCALE

描　　述：设置和查询载波周期比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的载波周期比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:CRI:SCALe <val>

:CONFigure:MATH:CRI:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_CRI_OFFSET

描　　述：设置和查询载波频率偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的载波周期偏置，范围【-1.0e8，1.0e8】，单位默认为 ns。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:CRI:OFFSet <val>

:CONFigure:MATH:CRI:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_PRF_SCALE

描　　述：设置和查询脉冲频率比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置脉冲频率比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PRF:SCALe <val>

:CONFigure:MATH:PRF:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_PRF_OFFSET

描　　述：设置脉冲频率偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲频率偏置，范围【-1.0e11，1.0e11】，单位默认为 Hz。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PRF:OFFSet <val>

:CONFigure:MATH:PRF:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_PRI_SCALE

描　　述：设置和查询脉冲周期比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲周期比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PRI:SCALe <val>

:CONFigure:MATH:PRI:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_PRI_OFFSET

描　　述：设置和查询脉冲周期偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲周期偏置，范围【-1.0e11，1.0e11】，单位默认为 ns。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PRI:OFFSet <val>

:CONFigure:MATH:PRI:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_PWID_SCALE

描　　述：设置和查询脉冲宽度比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲宽度比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PWID:SCALe <val>

:CONFigure:MATH:PWID:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_PWID_OFFSET

描　　述：设置的脉冲宽度偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲宽度偏置，范围【-1.0e11，1.0e11】，单位默认为 ns。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:PWID:OFFSet <val>

:CONFigure:MATH:PRI:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_OFFT_SCALE

描　　述：设置和查询脉冲空时间比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲空时间比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:OFFT:SCALe <val>

:CONFigure:MATH:OFFT:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_OFFT_OFFSET

描　　述：设置的脉冲空时间偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲空时间偏置，范围【-1.0e11，1.0e11】，单位默认为 ns。

ViConstString channelName：VI_NULL 或””

程控命令：:CONFigure:MATH:OFFT:OFFSet <val>

:CONFigure:MATH:OFFT:OFFSet?

AV3212D_ATTR_ CONFIGURE_MATH_DUTY_SCALE

描　　述：设置和查询脉冲占空比比例

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置脉冲占空比比例，范围【-20.0，20.0】。

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:MATH:DUTY:SCALe <val>

:CONFigure:MATH:DUTY:SCALe?

AV3212D_ATTR_ CONFIGURE_MATH_DUTY_OFFSET

描　　述：设置和查询脉冲占空比偏置

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的脉冲占空比偏置，范围【-100.0，100.0】，单位%。

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:MATH:DUTY:OFFSet <val>

:CONFigure:MATH:DUTY:OFFSet?

5.3.7 系统

AV3212D_ATTR_ TRIGGER_MODE

描　　述：设置和查询触发模式。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：3212D_VAL_TRIGGER_MODE_INT，内部门模式

3212D_VAL_TRIGGER_MODE_EXT，外部门模式

3212D_VAL_TRIGGER_MODE_ARM，ARM 模式

设置的触发模式索引号，0——内部门模式、1——外部门模式、2——ARM 模式。

ViConstString channelName：VI_NULL 或””

程控命令：

:TRIGger:MODE {INT|EXT|ARM}

:TRIGger:MODE ?

AV3212D_ATTR_ TRIGGER_DELAY

描　　述：设置和查询触发延时

访问函数：ViStatus AV3212D_SetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 value)

ViStatus AV3212D_GetAttributeViReal64 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViReal64 *value)

参数说明：ViReal64 value：设置的触发延时，范围【0，1.0e9】，单位默认为 ns。

ViConstString channelName：VI_NULL 或””

程控命令：:TRIGger:DELAy <val>

:TRIGger:DELAy?

AV3212D_ATTR_ SYSTEM_SCOPE_STATE

描　　述：设置和查询波形观察开关。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

ViStatus AV3212D_GetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean *value)

参数说明：　ViBoolean value：VI_FALSE，关

VI_TRUE，开

设置的波形观察开关索引号，0——关、1——开。

ViConstString channelName：VI_NULL 或””

程控命令：:SYSTem:SCOPe:STATe { OFF|ON|0|1}

:SYSTem:SCOPe:STATe?

AV3212D_ATTR_ CONFIGURE_DISPlAY_LANGUAGE

描　　述：设置和查询界面语言。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：

ViInt32 value：3212D_VAL_CONFIGURE_DISPLAY_LANGUAGE_CN，中文

3212D_VAL_CONFIGURE_DISPLAY_LANGUAGE_EN，英文

设置的语言状态索引号，0——中文、1——English。

ViConstString channelName：VI_NULL 或""

程控命令：:CONFigure:DISPlay:LANGuage {CN|EN}

:CONFigure:DISPlay:LANGuage?

AV3212D_ATTR_ SYSTEM_COMMUNICATE_GPIB_ADDRESS

描　　述：设置和查询 GPIB 地址。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：设置的 GPIB 地址，范围【0，30】。

ViConstString channelName：VI_NULL 或""

程控命令：:SYSTem:COMMunicate:GPIB:ADDRess val

:SYSTem:COMMunicate:GPIB:ADDRess?

AV3212D_ATTR_ SYSTEM_COMMUNICATE_LAN_IP

描　　述：设置和查询系统 ip 地址

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：待设置的 IP 地址

程控命令：:SYSTem:COMMunicate:LAN:IP <string>

:SYSTem:COMMunicate:LAN:IP？

AV3212D_ATTR_ SYSTEM_COMMUNICATE_LAN_ MASK

描　　述：设置和查询子网掩码

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：待设置的子网掩码

ViConstString channelName：VI_NULL 或""

程控命令：:SYSTem:COMMunicate:LAN:MASK <string>

:SYSTem:COMMunicate:LAN:MASK？

AV3212D_ATTR_ SYSTEM_COMMUNICATE_LAN_ GATEway

描　　述：设置和查询网关

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：待设置的网关

ViConstString channelName：VI_NULL 或””

程控命令：:SYSTem:COMMunicate:LAN:GATEway <string>

:SYSTem:COMMunicate:LAN:GATEway?

AV3212D_ATTR_ SYSTEM_COMMUNICATE_LAN_ DNS

描　　述：设置和查询 DNS

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：待设置的 DNS

ViConstString channelName：VI_NULL 或””

程控命令：:SYSTem:COMMunicate:LAN:DNS <string>

:SYSTem:COMMunicate:LAN:DNS?

AV3212D_ATTR_ CALIBRATION_INSERTION_START

描　　述：执行内插校准

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：NULL

ViConstString channelName：VI_NULL 或””

程控命令：:CALibration:INSertion:STARt

AV3212D_ATTR_ CALIBRATION_SAVE

描　　述：存储结果

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：NULL

ViConstString channelName：VI_NULL 或””

程控命令：:CALibration:SAVE

AV3212D_ATTR_ SYSTEM_SECURITY_KB

描　　述：禁止键盘，只可设置不可查询的命令。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

参数说明： ViBoolean value：VI_FALSE，关

ViConstString channelName：VI_NULL 或””

程控命令：:SYSTem:SECurity:KB { OFF|0}

AV3212D_ATTR_ SYSTEM_SECURITY_DISPLAY

描 述：关闭显示，只可设置不可查询的命令。

访问函数：ViStatus AV3212D_SetAttributeViBoolean (ViSession vi, ViConstString channelName, ViAttr attributeId, ViBoolean value)

参数说明： ViBoolean value：VI_FALSE，关

ViConstString channelName：VI_NULL 或””

程控命令：:SYSTem:SECurity:DISPlay { OFF|0}

5.3.8 通用命令函数

对于通用命令函数 ViConstString channelName 属性不是基于通道的，该参数指定为 VI_NULL 或空字符串。

AV3212D_ATTR_ID_QUERY_RESPONSE

描 述：返回 ID 查询响应字符串。可以通过在仪器初始化函数 3212D_init 或 3212D_InitWithOptions 中设置 IDQuery 参数为 VI_TRUE 获得。

访问函数：ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

参数说明：ViInt32 bufSize：缓冲区大小

ViChar value[]：ID 查询响应字符串。

程控命令：*IDN?

AV3212D_ATTR_IEEE_CLS

描 述：清除仪器状态寄存器和错误队列。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：VI_NULL 或””

程控命令：*CLS

AV3212D_ATTR_IEEE_ESE

描 述：设置或查询标准事件使能寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：取值范围， 0 ~ 255

ViInt32 *value：返回标准事件使能寄存器的值

程控命令：*ESE <data>

*ESE?

AV3212D_ATTR_IEEE_ESR

描　　述：查询标准事件状态寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回标准事件状态寄存器的值

程控命令：*ESR?

AV3212D_ATTR_IEEE_OPC

描　　述：设置标准事件状态寄存器的操作完成位。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：VI_NULL 或””

程控命令：*OPC

AV3212D_ATTR_IEEE_OPC_QUERY

描　　述：查询在指定的超时时间内所有异步操作是否完成。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：0，异步操作未完成

1，异步操作完成

程控命令：*OPC? [maxTimeoutMs]

AV3212D_ATTR_IEEE_RST

描　　述：仪器复位。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：VI_NULL 或””

程控命令：*RST

AV3212D_ATTR_IEEE_SRE

描　　述：设置或查询服务请求使能寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：取值范围：0 ~ 63 或 128 ~ 191

ViInt32 *value：返回服务请求使能寄存器的值

程控命令：*SRE <data>

*SRE?

AV3212D_ATTR_IEEE_STB_QUERY

描　　述：查询状态字节寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回的状态字节寄存器的值

程控命令：*STB?

AV3212D_ATTR_IEEE_WAI

描　　述：阻塞后续命令的执行等待异步操作完成。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：VI_NULL 或””

程控命令：*WAI

5.3.9 标准命令函数

对于标准命令函数 ViConstString channelName 属性不是基于通道的，该参数指定为 VI_NULL 或空字符串。

AV3212D_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION

描　　述：返回驱动兼容的类规范的主版本号。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回驱动兼容的类规范的主版本号

程控命令：无

AV3212D_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION

描　　述：返回驱动兼容的类规范的次版本号。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回驱动兼容的类规范的主版本号

程控命令：无

AV3212D_ATTR_SPECIFIC_DRIVER_DESCRIPTION

描　　述：特定驱动描述。

访问函数：ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

参数说明：ViInt32 bufSize：缓冲区大小

ViChar value[]：返回的特定驱动描述。

程控命令：无

AV3212D_ATTR_SPECIFIC_DRIVER_PREFIX

描　　述：仪器驱动前缀。

访问函数：ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

参数说明：ViInt32 bufSize：缓冲区大小

ViChar value[]：返回的仪器驱动前缀。

程控命令：无

AV3212D_ATTR_SPECIFIC_DRIVER_REVISION

描　　述：仪器驱动版本号。

访问函数：ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

参数说明：ViInt32 bufSize：缓冲区大小

ViChar value[]：返回的仪器驱动版本号。

程控命令：无

AV3212D_ATTR_SPECIFIC_DRIVER_VENDOR

描　　述：　驱动开发商。

访问函数：ViStatus AV3212D_GetAttributeViString (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 bufSize, ViChar value[])

参数说明：ViInt32 bufSize：缓冲区大小

ViChar value[]：返回驱动开发商。

程控命令：无

AV3212D_ATTR_STATUS_OPERATION_CONDITION

描　　述：返回操作状态寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回操作状态寄存器的值

程控命令：:STATus:OPERation:CONDition?

AV3212D_ATTR_STATUS_OPERATION_ENABLE

描　　述：设置或查询操作状态使能寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的操作状态使能寄存器的值

ViInt32 *value：返回的操作状态使能寄存器的值

程控命令：:STATus:OPERation:ENABle <enable>

:STATus:OPERation:ENABle?

AV3212D_ATTR_STATUS_OPERATION_EVENT

描　　述：查询操作事件寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回的操作事件寄存器的值

程控命令：:STATus:OPERation[:EVENt]?

AV3212D_ATTR_STATUS_OPERATION_NTRANSITION

描　　述：设置或查询操作状态寄存器中的负向转换寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的负向转换寄存器的值

ViInt32 *value：返回的负向转换寄存器的值

程控命令：:STATus:OPERation:NTRansition <integer>

:STATus:OPERation:NTRansition?

AV3212D_ATTR_STATUS_OPERATION_PTRANSITION

描　　述：设置或查询操作状态寄存器中的正向转换寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,

ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的正向转换寄存器的值

ViInt32 *value：返回的正向转换寄存器的值

程控命令：:STATus:OPERation:PTRansition <integer>

:STATus:OPERation:PTRansition?

AV3212D_ATTR_STATUS_PRESET

描　　述：将状态寄存器恢复到一个可知的状态。

访问函数：ViStatus AV3212D_SetAttributeViString (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViConstString value)

参数说明：ViConstString value：VI_NULL 或""

程控命令：:STATus:PRESet

AV3212D_ATTR_STATUS_QUESTIONABLE_CONDITION

描　　述：查询可疑状态寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回可疑状态寄存器的值

程控命令：:STATus:QUEStionable:CONDition?

AV3212D_ATTR_STATUS_QUESTIONABLE_ENABLE

描　　述：设置或查询可疑状态使能寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的可疑状态使能寄存器的值

ViInt32 *value：返回的可疑状态使能寄存器的值

程控命令：:STATus:QUEStionable:ENABle <enable>

:STATus:QUEStionable:ENABle?

AV3212D_ATTR_STATUS_QUESTIONABLE_EVENT

描　　述：查询可疑状态事件寄存器的值。

访问函数：ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName,
ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 *value：返回的操作事件寄存器的值

程控命令：:STATus:QUEStionable [:EVENt]?

AV3212D_ATTR_STATUS_QUESTIONABLE_NTRANSITION

描　　述：设置或查询可疑状态寄存器中的负向转换寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的负向转换寄存器的值

ViInt32 *value：返回的负向转换寄存器的值

程控命令：:STATus:QUEStionable:NTRansition <integer>

:STATus:QUEStionable:NTRansition?

AV3212D_ATTR_STATUS_QUESTIONABLE_PTRANSITION

描　　述：设置或查询可疑状态寄存器中的正向转换寄存器的值。

访问函数：ViStatus AV3212D_SetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 value)

ViStatus AV3212D_GetAttributeViInt32 (ViSession vi, ViConstString channelName, ViAttr attributeId, ViInt32 *value)

参数说明：ViInt32 value：待设置的正向转换寄存器的值

ViInt32 *value：返回的正向转换寄存器的值

程控命令：:STATus:QUEStionable:PTRansition <integer>

第六章 编程示例

这些程控示例都是使用不同的 I/O 库和程序设计语言来说明对宽带微波毫米波脉冲频率计的控制，并且对设备的控制都是通过 GPIB 和 LAN 接口来实现通信的。本章中的示例许多都是交互式的，使用者将被提示执行确定的动作或者检查频率计的操作或其功能性。

第一节 设计示例运行环境

6.1.1 C/C++的示例

本章所描述的编程示例，已在如下配置的计算机上运行成功。

- IBM 兼容、奔腾级以上的 PC 机
- Windows 2000 或 Windows XP 操作系统
- VC6.0 或者 Visual studio 2005 集成开发环境
- NI 公司的 PCI-GPIB 接口卡或 Agilent 公司的 GPIB 接口卡
- NI 的 VISA 库或者 Agilent 的 VISA 库
- 安装 IVI Shared Components 和 NI VISA 或 Agilent VISA 库。可以通过安装 Agilent IO Libraries 进行上述功能的安装。
- 网卡

6.1.2 运行 C/C++设计程序

运行 C/C++编写的程序示例，您必须在 VC6.0 或者 Visual studio 2005 的项目中包含必须的文件
如果您使用 VISA 库必须进行以下步骤：

- 添加 visa32.lib 到源文件
- 添加 visa.h 到头文件

注意 **1、关于 VISA 库的更详细资料请分别在 NI 公司和 Agilent 公司的网站查阅。**

第二节 GPIB 设计示例

6.2.1 使用示例之前

如果您使用 Agilent 公司的 GPIB 接口卡，那么您必须正确的安装 Agilent 的 VISA 库；同样如果使用 NI 公司的 PCI-GPIB 接口卡，您必须也要正确的安装 NI-488.2 库。

6.2.2 GPIB 接口卡设计示例

6.2.2.1 使用 VISA 库和 C 语言来实现接口检查

```
/*****************************************************************/
```

本例检查了 GPIB 的连接和接口的功能，启动 VC6.0，添加必须的文件，把下述代码输入您的.cpp 文件

```
/*****************************************************************/
#include <visa.h>
#include <stdio.h>
#include “stdAfx.h”
#include <stdlib.h>
void main()
{
   ViSession defaultRM,vi;//声明一个类型为 VISession 的变量
   char buf [256] = {0};
ViStatus vistatus = 0;
   vistatus = viOpenDefaultRM(&defaultRM);
//打开 GPIB 任务，地址为 19
   vistatus=viOpen(defaultRM,"GPIB::19::INSTR",VI_NULL,VI_NULL,&vi);
   if(vistatus)
   {
         printf(“任务无法打开，请重新检查设备并连接\n”);
         exit(0);
   }
   viPrintf(vi,"*RST\n");//初始化频率计
```

```
    printf(“频率计处于远控状态，并显示远控标志\n”);
    viPrintf(vi,"*IDN?\n");
viScanf(vi,"%t",buf);
printf("Instrument Identification String:%s\n",buf);
viClose(vi);
viClose(defaultRM);
}
```

6.2.2.2 使用 VISA 库和 C 语言来实现查询功能

```
/*****************************************************************/
```

本例使用 VISA 库的功能，查询了设备的不同状态和条件，启动 VC6.0，添加必须的文件，把下述代码输入您的.cpp 文件

```
/*****************************************************************/
#include <visa.h>
#include “stdAfx.h”
#include <iostream>
#include <conio.h>
#include <stdlib.h>
void main()
{
    ViSession defaultRM,vi; //声明类型为 VISession 的变量
ViStatus vistatus = 0;
    Char buff[256]； //声明存储字符数据的数组
    vistatus = viOpenDefaultRM(&defaultRM);//打开 GPIB 任务，地址为 19
    vistatus=viOpen(defaultRM,"GPIB::19::INSTR",VI_NULL,VI_NULL,&vi);
    if(vistatus)
    {
          printf(“任务无法打开，请重新检查设备并连接\n”);
          exit(0);
    }
    viPrintf(vi，”*RST\n”); //复位频率计
```

```
    viPrintf(vi，”:CONFigure:MEASure:MODE?\n”);//查询测量类型
    viScanf(vi，"%t"，buff); //把查询结果放入数组
    printf(“MEASure MODE IS: %s\n”，buff);//显示测量类型
    viPrintf(vi，”:CONFigure:MEASure:PULSe:MODE?\n”); //查询脉冲测量模式
    viScanf(vi，"%t"，buff)； //把查询结果放入数组
    printf(“PULse MODE is: %s\n”，buff) //显示脉冲测量模式
    viPrintf(vi，”:CONFigure:MEASure:TYPE?\n”); //查询测量类型
    viScanf(vi，"%t"，buff)； //把查询结果放入数组
    printf(“source measure type is: %s\n”，buff); //显示频率模式
    viclose(vi);
    viclose(defaultRM);
}
```

6.2.2.3 使用 VISA 库和 C 语言测量载波频率

```
/*****************************************************************/
```

本例使用 VISA 库的功能，设置频率计输出 500MHz 信号的点频和-2dBm 的功率，并查询当前频率和功率，启动 VC6.0，添加必须的文件，把下述代码输入您的.cpp 文件

```
/*****************************************************************/
#include “stdAfx.h”
#include <visa.h>
#include <iostream>
#include <stdlib.h>
#include <conio.h>
void main()
{
    ViSession defaultRM,vi; //声明类型为 VISession 的变量
    ViStatus vistatus = 0;            //为了设备通信
    char buff[256] = {0};                  //声明存储字符数据的变量
    int num;                        //声明存储整形数据的变量
//打开 GPIB 任务，地址为 19
    vistatus = viOpenDefaultRM(&defaultRM);
    vistatus=viOpen(defaultRM,"GPIB::1::INSTR",VI_NULL,VI_NULL,&vi);
```

```
        if(vistatus)
        {
              printf(“任务无法打开，请重新检查设备并连接\n”);
              exit(0);
        }
        viPrintf(vi,"*RST\n");              //复位频率计
        viPrintf(vi，”:CONFigure:MEASure:MODE CW\n”);//设置测量类型连续波
        viPrintf(vi，” :CONFigure:MEASure:MODE?\n”); //查询测量类型
        viScanf(vi，"%t"，buff)； //把查询结果放入数组
        viScanf(vi,"%t",buff);  //显示测量类型
        viPrintf(vi，”:CONFigure:FREQuency (@1)\n”);//设置通道 1 波
        viPrintf(vi，”:CONFigure:FREQuency?\n”); /查询测量通道
        viScanf(vi，"%t"，buff) ;              //把查询结果放入数组
        printf(“measure FREQuency: %s\n”，buff); //显示测量类型
        viPrintf(vi，” :CONFigure:MEASure:TYPE CRF\n”);//设置测量参数 CRF
        viPrintf(vi，”:CONFigure:MEASure:TYPE?\n”); /查询测量类型
        viScanf(vi，"%t"，buff);              //把查询结果放入数组
        printf(“measure type: %s\n”，buff); //显示测量类型
        viPrintf(vi，”:FETCh:CARRier:FREQuency?\n”); /查询 CRF
        viScanf (vi，"%t"，buff);              //把查询结果放入数组
        printf(“measure FREQuency: %s\n”，buff); //显示 CRF
        viClose(vi);
        viClose(defaultRM);
    }
}
```

第三节　网络设计示例

6.3.1 使用示例之前

为了能正确使用以下示例，您必须将您的主机地址与频率计的 IP 地址相匹配。(本手册网络设计示例在 VC6.0 下利用 WINSOCK 组件建立 socket 实现)

6.3.2 网络设计示例

6.3.2.1 使用 socket 和 C++实现查询载波频率

```
#include "stdafx.h"
#include <afxsock.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

void main()
{
    char recvbuff[100];
    int  nPort = 5000;
    int  nErr;
    ZeroMemory(recvbuff,100);
    /*初始化 socket 环境*/
    WSADATA wsd;
    WSAStartup(MAKEWORD(1,1),&wsd);
   /*创建 TCP 套接字*/
   SOCKET LocalSocket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
   if (LocalSocket == SOCKET_ERROR)
   {
      AfxMessageBox("创建套接字失败",MB_OK);
      return ;
   }
```

```
   struct sockaddr_in addr;
   addr.sin_family = AF_INET;
/*设置 IP 地址和端口号*/
   addr.sin_port = htons(nPort);
   addr.sin_addr.s_addr=inet_addr("172.141.8.201");
   nErr = connect(LocalSocket,(struct sockaddr *)&addr,sizeof(addr));

   if (nErr == SOCKET_ERROR)
    {
      AfxMessageBox("连接失败",MB_OK);
      return ;
    }
 /*查询 CRF 值*/
nErr = send(LocalSocket," :FETCh:CARRier:FREQuency?\n",12,0);
if (nErr == SOCKET_ERROR)
    {
          AfxMessageBox("发送数据失败",MB_OK);
          closesocket(LocalSocket);
          return ;
    }
/*将查询结果放入数组*/
recv(LocalSocket,recvbuff,100,0);
cout<<"显示载波频率值:"<<recvbuff<<endl;
/*关闭 socket 释放资源*/
   closesocket(LocalSocket);
   WSACleanup();
}
```

6.3.2.2 Sockets LAN 编程和 C

在“使用 Sockets 为 LAN 查询”列出的程序由 lanio.c 和 getopt.c 两个文件组成。lanio.c 文件有两个 main 函数：int main()和 int main1()。

int main() 函数允许从命令行和频率计进行交互式通信。程序从命令行通过 SCPI 命令读频率计的主机名。随后对频率计打开 5000 socket 端口，然后发送命令。如果命令被查询，程序做频率计响应查询，随后打印响应。

int main1() 被改名为 int main()将会给频率计输出一列命令。可以使用该格式作为模板添加到自己的代码中。

Windows Sockets

在 Windows 中，因为 fread() 和 fwrite() 不会在 sockets 上工作，故必须使用程序 send() 和 recv()。下面说明了在 Microsoft Visual C++ 6.0 环境下运行交互式程序的步骤：

1. 将 lanio.c 改名为 lanio.cpp，getopt.c 改名为 getopt.cpp。将它们添加到 Visual C++ 工程的源文件夹中。

注意 **lanio.cpp 文件中的 int main() 函数通过用户键入 SCPI 命令允许以一行一行的给频率计发送命令。使用 int main1(0) 函数能够以“程序格式”输出一系列命令。参见下面的使用 main1()函数编程。**

2. 在 **Build** 菜单选在 **Rebuild All**。然后点击 **Execute Lanio.exe**。将出现 Debug window 提示 “Press any key to continue.” 说明程序已经编译能被用来给频率计发送命令。

3. 点击 Start, Programs, Command Prompt. 将会出现命令提示符窗口。

4. 在命令提示符下进入包含 lanio.exe 文件的目录，然后进入 Debug 文件夹。例如：C:\SocketIO\Lanio\Debug.

5. 进入包含 lanio.exe 文件的目录后，在命令提示符下敲入如下命令：lanio xxxxx “*IDN?”。例如：C:\SocketIO\Lanio\Debug>lanio xxxxx “*IDN?” xxxxx 是频率计的 IP 地址。在命令行提示符下以一行一行的格式给频率计输出 SCPI 命令。

6. 在命令行提示符下键入 exit 退出程序。

6.3.2.3 使用 Sockets 为 Lan 查询

lanio.c 和 getopt.c 完成如下功能：

- ◆ 在端口 5001 建立 TCP/IP 连接
- ◆ 使用结果文件描述符与设备进行会话的使用常规 socket I/O 机制
- ◆ 出错检查
- ◆ 设置测量类型连续波
- ◆ 设置测量通道为通道 2
- ◆ 设置测量测量参数 CRF
- ◆ 查询载波频率

```
/*****************************************************************
*程序名：lanio.c
*功能：通过 TCP/IP 使用命令行参数与频率计会话。和 5000 端口建立 TCP/IP 连接，
```

```
*结果文件描述符用来与使用常规 socket I/O 机制的设备会话。$
*例如：
*查询频率计频率：lanio xx.xxx.xx.x 'FREQ?'
*检查错误（得到一个错误）：lanio xx.xxx.xx.x 'syst:err?'
*从一个文件发送一系列命令，并对其计数：cat scpi_cmds | lanio -n xx.xxx.xx.x
*************************************************************/
/*************************************************************
* Header: lanio.c 04/24/01
*************************************************************/
#ifndef _WIN32 /* Visual C++ 6.0 will define this */
# define WINSOCK
#endif

#include <stdio.h> /* for fprintf and NULL */
#include <string.h> /* for memcpy and memset */
#include <stdlib.h> /* for malloc(), atol() */
#include <errno.h> /* for strerror */

#ifdef WINSOCK
#include <windows.h>
#ifndef _WINSOCKAPI_
#include <winsock.h> // BSD-style socket functions
#endif
#endif

#ifdef WINSOCK
/* Declared in getopt.c.*/
extern char *optarg;
extern int optind;
extern int getopt(int argc, char * const argv[], const char* optstring);
#else
```

```
#include <unistd.h> /* for getopt(3C) */
#endif

#define COMMAND_ERROR (1)
#define NO_CMD_ERROR (0)
#define SCPI_PORT 5000
#define INPUT_BUF_SIZE (64*1024)
/*****************************************************************
* Display usage
*****************************************************************/
static void usage(char *basename)
{
   fprintf(stderr,"Usage: %s [-nqu] <ip_addr> [<command>]\n",
basename);
   fprintf(stderr," %s [-nqu] <ip_addr> < stdin\n", basename);
   fprintf(stderr," -n, number output lines\n");
   fprintf(stderr," -q, quiet; do NOT echo lines\n");
   fprintf(stderr," -e, show messages in error queue when done\n");
}

#ifdef WINSOCK
int init_winsock(void)
{
   WORD wVersionRequested;
   WSADATA wsaData;
   int err;
   wVersionRequested = MAKEWORD(1, 1);
   wVersionRequested = MAKEWORD(2, 0);
   err = WSAStartup(wVersionRequested, &wsaData);
   if (err != 0)
   {
         /* Tell the user that we couldn't find a useable */
```

```
            /* winsock.dll. */
            fprintf(stderr, "Cannot initialize Winsock 1.1.\n");
            return -1;
        }
        return 0;
    }
    int close_winsock(void)
    {
        WSACleanup();
        return 0;
    }

    #endif /* WINSOCK */

    SOCKET openSocket(const char * ip_addr, int portNumber)
    {
        struct hostent *hostPtr;
        struct sockaddr_in peeraddr_in;
        SOCKET s;
        memset(&peeraddr_in, 0, sizeof(struct sockaddr_in));
        /******************/
        /* create a socket */
        /******************/
        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s == INVALID_SOCKET)
        {
             fprintf(stderr,"unable to create socket to '%s': %s\n",ip_addr,
strerror(errno));
            return INVALID_SOCKET;
        }
        peeraddr_in.sin_addr.S_un.S_addr=inet_addr(ip_addr);
        peeraddr_in.sin_family = AF_INET;
```

```
       peeraddr_in.sin_port = htons((unsigned short)portNumber);
       if (connect(s, (const struct sockaddr*) &peeraddr_in, sizeof (struct
sockaddr_in)) == SOCKET_ERROR)
       {
            fprintf(stderr,"unable to create socket to '%s': %s\n",ip_addr,
strerror(errno));
             return INVALID_SOCKET;
       }
       return s;
   }
   /*************************************************************
   > $Function: commandInstrument$
   *************************************************************/
   int commandInstrument(SOCKET sock,const char *command)
   {
       int count;
       /* fprintf(stderr, "Sending \"%s\".\n", command); */
       if (strchr(command, '\n') == NULL)
       {
             fprintf(stderr, "Warning: missing newline on command %s.\n",
command);
       }
       count = send(sock, command, strlen(command), 0);
       if (count == SOCKET_ERROR)
       {
             return COMMAND_ERROR;
       }
       return NO_CMD_ERROR;
   }
   /*************************************************************
   * recv_line(): similar to fgets(), but uses recv()
   *************************************************************/
```

```
char * recv_line(SOCKET sock, char * result, int maxLength)
{
#ifdef WINSOCK
    int cur_length = 0;
    int count;
    char * ptr = result;
    int err = 1;
    while (cur_length < maxLength)
    {
          /* Get a byte into ptr */
          count = recv(sock, ptr, 1, 0);
          /* If no chars to read, stop. */
          if (count < 1)
          {
          break;
          }
          cur_length += count;
          /* If we hit a newline, stop. */
          if (*ptr == '\n')
          {
          ptr++;
          err = 0;
          break;
          }
          ptr++;
    }
    *ptr = '\0';
    if (err)
    {
          return NULL;
    }
```

```
       else
       {
             return result;
       }
   #else
   /*************************************************************
   > $Function: queryInstrument$
   *************************************************************/
   long queryInstrument(SOCKET sock,const char *command, char *result,
size_t maxLength)
   {
       long ch;
       char tmp_buf[8];
       long resultBytes = 0;
       int command_err;
       int count;
       /*********************************************************
       * Send command to signal generator
       *********************************************************/
       command_err = commandInstrument(sock, command);
       if (command_err)
             return COMMAND_ERROR;
       /*********************************************************
             * Read response from signal generator
       *********************************************************/
       count = recv(sock, tmp_buf, 1, 0); /* read 1 char */
       ch = tmp_buf[0];
       if ((count < 1) || (ch == EOF) || (ch == '\n'))
       {
             *result = '\0'; /* null terminate result for ascii */
             return 0;
       }
```

```
/* use a do-while so we can break out */
do
{
      if (ch == '#')
      {
      /* binary data encountered - figure out what it is */
      long numDigits;
      long numBytes = 0;
      /* char length[10]; */
      count = recv(sock, tmp_buf, 1, 0); /* read 1 char */
      ch = tmp_buf[0];
      if ((count < 1) || (ch == EOF))
            break; /* End of file */
      if (ch < '0' || ch > '9')
            break; /* unexpected char */
            numDigits = ch - '0';
      if (numDigits)
      {
      /* read numDigits bytes into result string. */
      count = recv(sock, result, (int)numDigits, 0);
      result[count] = 0; /* null terminate */
      numBytes = atol(result);
      }
      if (numBytes)
      {
      resultBytes = 0;
      /* Loop until we get all the bytes we requested. */
      /* Each call seems to return up to 1457 bytes */
      do
      {
      int rcount;
```

```
            rcount = recv(sock, result, (int)numBytes, 0);
            resultBytes += rcount;
            result += rcount; /* Advance pointer */
            }
            while ( resultBytes < numBytes );

      /*************************************************************
   * For LAN dumps, there is always an extra trailing
   * newline Since there is no EOI line. For ASCII dumps
   * this is great but for binary dumps, it is not needed.
      *************************************************************/
      if (resultBytes == numBytes)
      {
      char junk;
      count = recv(sock, &junk, 1, 0);
      }
      }
      else
      {
   /* indefinite block ... dump til we can an extra line feed */
      do
      {
      if (recv_line(sock, result, maxLength) == NULL)
      break;
      if (strlen(result)==1 && *result == '\n')
            break;
      resultBytes += strlen(result);
      result += strlen(result);
      }
      while (1);
   }
      }
```

```
            else
            {
                  /* ASCII response (not a binary block) */
                  *result = (char)ch;
                  if (recv_line(sock, result+1, maxLength-1) == NULL)
                      return 0;
                  /* REMOVE trailing newline, if present. And terminate
string. */
                  resultBytes = strlen(result);
                  if (result[resultBytes-1] == '\n')
                      resultBytes -= 1;
                  result[resultBytes] = '\0';
            }
      }
      while (0);
      return resultBytes;
   }
   /*****************************************************************
   > $Function: showErrors$
   *****************************************************************/
   void showErrors(SOCKET sock)
   {
      const char * command = "SYST:ERR?\n";
      char result_str[256];
      do
      {
      queryInstrument(sock, command, result_str, sizeof(result_str)-1);

      /*************************************************************
            * Typical result_str:
            * -221,"Settings conflict; Frequency span reduced."
            * +0,"No error"
```

```
            * Don't bother decoding.

        ************************************************************/
            if (strncmp(result_str, "+0,", 3) == 0)
            {
                /* Matched +0,"No error" */
                break;
            }
            puts(result_str);
        }
        while (1);
    }
    /*****************************************************************
    > $Function: isQuery$
    *****************************************************************/
    unsigned char isQuery( char* cmd )
    {
        unsigned char q = 0 ;
        char *query ;
        /*********************************************************/
        /* if the command has a '?' in it, use queryInstrument. */
        /* otherwise, simply send the command. */
        /* Actually, we must be a more specific so that */
        /* marker value querys are treated as commands. */
        /* Example: SENS:FREQ:CENT (CALC1:MARK1:X?) */
        /*********************************************************/
        if ( (query = strchr(cmd,'?')) != NULL)
        {
        /* Make sure we don't have a marker value query, or
        * any command with a '?' followed by a ')' character.
        * This kind of command is not a query from our point of view.
        * The signal generator does the query internally, and uses the result.
```

```
    */
          query++ ; /* bump past '?' */
          while (*query)
          {
          if (*query == ' ') /* attempt to ignore white spc */
          query++ ;
          else
          break ;
          }
          if ( *query != ')' )
          {
          q = 1 ;
          }
    }
    return q ;
}
////////////////////////////////////////////////////////////////
int main(int argc, char *argv[])
{
    SOCKET instSock;
    char *charBuf = (char *) malloc(INPUT_BUF_SIZE);
    char *basename;
    int chr;
    char command[1024];
    char *destination;
    unsigned char quiet = 0;
    unsigned char show_errs = 0;
    int number = 0;
    basename = strrchr(argv[0], '/');
    if (basename != NULL)
          basename++ ;
```

```
    else
          basename = argv[0];
    while((chr=getopt(argc,argv,"qune"))!=EOF)
    {
          switch(chr)
          {
          case 'q':
                quiet =1;break;
          case 'n':
                number =1; break ;
          case 'e':
                show_errs=1; break ;
          case 'u':
          case '?':
                usage(basename); exit(1) ;
          }
    }//end while
    /* now look for ip_addr and optional <command>*/
    if (optind < argc)
    {
          destination = argv[optind++] ;
          strcpy(command, "");
          if (optind < argc)
          {
                while (optind < argc)
{
/* <ip_addr> <command> provided; only one command string */
strcat(command, argv[optind++]);
if (optind < argc)
{
    strcat(command, " ");
```

```
}
else
   {
   strcat(command, "\n");
               }
         }
         }
         else
         {
               /*Only <ip_addr> provided; input on <stdin> */
               strcpy(command, "");
               if (optind > argc)
               {
               usage(basename);
               exit(1);
               }
         }
   }
   else
   {
         /* no ip_addr! */
         usage(basename);
         exit(1);
   }
   /*******************************************************
   /* open a socket connection to the instrument
   /*******************************************************/
#ifdef WINSOCK
   if (init_winsock() != 0)
   {
         exit(1);
```

```
        }
    #endif /* WINSOCK */
        instSock = openSocket(destination, SCPI_PORT);
        if (instSock == INVALID_SOCKET)
        {
              fprintf(stderr, "Unable to open socket.\n");
              return 1;
        }
        /* fprintf(stderr, "Socket opened.\n"); */
        if (strlen(command) > 0)
        {
        /*******************************************************
              /* if the command has a '?' in it, use queryInstrument. */
              /* otherwise, simply send the command. */
              /*******************************************************/
              if ( isQuery(command) )
              {
                    long bufBytes;
                    bufBytes = queryInstrument(instSock, command,charBuf,
INPUT_BUF_SIZE);
                    if (!quiet)
                    {
                        fwrite(charBuf, bufBytes, 1, stdout);
                        fwrite("\n", 1, 1, stdout) ;
                        fflush(stdout);
                    }
              }
              else
              {
                    commandInstrument(instSock, command);
              }
        }
```

```
      else
      {
            /* read a line from <stdin> */
            while ( gets(charBuf) != NULL )
            {
            if ( !strlen(charBuf) )
            continue ;
            if ( *charBuf == '#' || *charBuf == '!' )
            continue ;
            strcat(charBuf, "\n");
            if (!quiet)
            {
            if (number)
            {
            char num[10];
      sprintf(num,"%d: ",number);
      fwrite(num, strlen(num), 1, stdout);
            }
            fwrite(charBuf, strlen(charBuf), 1, stdout) ;
            fflush(stdout);
            }
            if ( isQuery(charBuf) )
            {
            long bufBytes;
            /* Put the query response into the same buffer as the*/
            /* command string appended after the null terminator.*/

      bufBytes=queryInstrument(instSock,charBuf,charBuf+strlen(charBuf)
+ 1,INPUT_BUF_SIZE -strlen(charBuf) );
            if (!quiet)
            {
            fwrite(" ", 2, 1, stdout) ;
```

```
        fwrite(charBuf+strlen(charBuf)+1,bufBytes,1,stdout);
              fwrite("\n", 1, 1, stdout) ;
              fflush(stdout);
              }
                    }
                    else
                    {
                         commandInstrument(instSock, charBuf);
                    }
                    if (number)
                         number++;
              }
        }
        if (show_errs)
        {
              showErrors(instSock);
        }
    #ifdef WINSOCK
        closesocket(instSock);
        close_winsock();
    #else
        close(instSock);
    #endif /* WINSOCK */
        return 0;
    }
    /*******************************************************************/
    /* $Function: main1$ */
    /*******************************************************************/
    int main1()
    {
        SOCKET instSock;
```

```
        long bufBytes;
        char *charBuf = (char *) malloc(INPUT_BUF_SIZE);
        /*****************************************/
        /* open a socket connection to the instrument*/
        /*****************************************/
    #ifdef WINSOCK
        if (init_winsock() != 0)
        {
              exit(1);
        }
    #endif /* WINSOCK */
        instSock = openSocket("xxxxxx", SCPI_PORT); /* Put your ip_addr here
*/
        if (instSock == INVALID_SOCKET)
        {
              fprintf(stderr, "Unable to open socket.\n");
              return 1;
        }
        /* fprintf(stderr, "Socket opened.\n"); */
        bufBytes = queryInstrument(instSock, "*IDN?\n", charBuf,
INPUT_BUF_SIZE);
        printf("ID: %s\n",charBuf);
        commandInstrument(instSock, " :CONFigure:MEASure:MODE CW\n");
        printf("\n");
        commandInstrument(instSock, " :CONFigure:FREQuency (@2) \n");
    printf("\n");
    commandInstrument(instSock, " :CONFigure:MEASure:TYPE CRF \n");
    printf("\n");
        bufBytes=queryInstrument(instSock," :FETCh:CARRier:FREQuency\n",
charBuf, INPUT_BUF_SIZE);
        printf("CRF value is: %s\n",charBuf);
        printf("\n");
```

```
#ifdef WINSOCK
   closesocket(instSock);
   close_winsock();
#else
   close(instSock);
#endif /* WINSOCK */
   return 0;
}
/*****************************************************************
程序文件名：getopt.c
*****************************************************************/
#include <stdio.h> /* For NULL, EOF */
#include <string.h> /* For strchr() */
char *optarg; /* Global argument pointer. */
int optind = 0; /* Global argv index. */
static char *scan = NULL; /* Private scan pointer. */
int getopt( int argc, char * const argv[], const char* optstring)
{
   char c;
   char *posn;
   optarg = NULL;
   if (scan == NULL || *scan == '\0')
   {
         if (optind == 0)
               optind++;
         if (optind >= argc || argv[optind][0] != '-' || argv[optind][1]
== '\0')
               return(EOF);
         if (strcmp(argv[optind], "--")==0)
         {
               optind++;
               return(EOF);
```

```
            }
            scan = argv[optind]+1;
            optind++;
        }
        c = *scan++;
        posn = strchr(optstring, c); /* DDP */
        if (posn == NULL || c == ':')
        {
            fprintf(stderr, "%s: unknown option -%c\n", argv[0], c);
            return('?');
        }
        posn++;
        if (*posn == ':')
        {
            if (*scan != '\0')
            {
                optarg = scan;
                scan = NULL;
            }
            else
            {
                optarg = argv[optind];
                optind++;
            }
        }
        return(c);
    }
```

第四节 驱动库设计示例

6.4.1 在 LabWindows CVI 9.0 中的使用方法

在使用 IVI-C 仪器驱动库时，必须包含“3212D.h”头文件和正确导入“3212D.lib”库文件。下面介绍该驱动库在 LabWindows CVI 9.0 中的使用步骤：

（1） 从“Edit”菜单选择“Add File to Project…”，然后选择“Instrument(.fp)”；

（2） 将头文件“3212D.h”、库文件“3212D.lib”和面板文件“3212D.fp”文件导入到测试项目中；

（3） 在测试项目中添加使用的头文件，如“#include “3212D.h”” 文件；

（4） 按上述步骤就可以使用驱动库了。

6.4.1.1 仪器驱动库初始化和销毁

```
/*****************************************************************/
```

本例检查了仪器驱动库的连接和接口功能，启动 LabWindows CVI 9.0，添加必须的文件，把下述代码输入您的.c 文件

```
/*****************************************************************/
#include “3212D.h”
```

static ViSession session = VI_NULL;

```
void main()
{
   ViStatus status;
   // Initialize the driver using a logical name
   status = 3212D_InitWithOptions("MyLogicalName", VI_TRUE, VI_TRUE,
"", &session);
   // Close the session
   status = 3212D_close(session);
}
```

6.4.1.2 启动载波频率测量、等待测量完成并读取测量结果

/***/

本例执行载波频率测量、等待测量完成并读取测量平均值和标准差，启动 LabWindows CVI 9.0，添加必须的文件，把下述代码输入您的.c 文件

```
/*****************************************************************/
#include “3212D.h”
```

```
static ViSession session = VI_NULL;

void main()

{

   ViStatus status;

   ViReal64 average = 0.0;

   // Initialize the driver using a logical name

   status = 3212D_InitWithOptions("MyLogicalName", VI_TRUE, VI_TRUE,
"", &session);

   status = 3212D_SetAttributeViInt32 (session, NULL,
          AV3212D_ATTR_CONFIGURE_MEASURE_MODE,

          3212D_VAL_CONFIGURE_MEASURE_MODE_CW);

   status = 3212D_SetAttributeViInt32 (session, NULL,
          AV3212D_ATTR_CONFIGURE_MEASURE_TYPE,

          3212D_VAL_CONFIGURE_MEASURE_TYPE_CRF);

   status = 3212D_SetAttributeViInt32 (session, NULL,
          AV3212D_ATTR_CONFIGURE_FREQUENCY,

          3212D_VAL_CONFIGURE_FREQUENCY_CH2);


   ViStatus AV3212D_GetAttributeViReal64 (session,NULL,
          AV3212D_ATTR_FETCH_CARRIER_FREQUENCY, &average);


   printf(“CRF value is：%lf”, average);

   // Close the session

   status = 3212D_close(session);

}
```

6.4.2 在 Visual C++ 2005 中的使用方法

在使用 IVI-C 仪器驱动库时，必须包含“3212D.h”头文件和正确导入“3212D.lib”库文件。下面介绍该驱动库在 Visual C++2005 中的使用步骤：

（1） 选择“项目”菜单选择“属性”菜单，然后选择“C/C++”-〉“常规”-〉“附加包含目录”选项，添加编译所需要的头文件所在路径；

（2） 选择“项目”菜单选择“属性”菜单，然后选择“连接器”-〉“常规”-〉“附加库目录”选项，添加链接所需要的库文件所在路径；

（3） 选择“项目”菜单选择“属性”菜单，然后选择“连接器”-〉“输入”-〉“附加依赖项”选项，输入“3212D.lib”添加项目运行时的库文件，也可以通过在测试项目顶部以“#pragma comment(lib,"3212D")”方式添加项目运行时的库文件；

（4） 在测试项目中添加使用的头文件，如“#include “3212D.h”” 文件；

（5） 按上述步骤就可以使用驱动库了。

6.4.2.1 仪器驱动库初始化和销毁

```
/****************************************************************/
```

本例检查了仪器驱动库的连接和接口功能，启动 Visual C++2005，添加必须的文件，把下述代码输入您的.cpp 文件

```
/****************************************************************/
#include "stdafx.h"
#include “3212D.h”
#pragma comment(lib,"3212D")
static ViSession session = VI_NULL;
int _tmain(int argc, _TCHAR* argv[])
{
   ViStatus status;
   // Initialize the driver using a logical name
   status = 3212D_InitWithOptions("MyLogicalName", VI_TRUE, VI_TRUE,
"", &session);
   // Close the session
   status = 3212D_close(session);
}
```

6.4.2.2 启动载波频率测量并读取测量结果

/**/

本例执行载波频率测量并读取测量结果，启动 Visual C++2005，添加必须的文件，把下述代码输入您的.cpp 文件

```
/****************************************************************/
#include "stdafx.h"
#include “3212D.h”
#pragma comment(lib,"3212D")
static ViSession session = VI_NULL;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
   ViStatus status;
   ViInt32 status_ofMeasurement = 0;
   ViReal64 average = 0.0;
   // Initialize the driver using a logical name
   status = 3212D_InitWithOptions("MyLogicalName", VI_TRUE, VI_TRUE,
"", &session);
   status = 3212D_SetAttributeViInt32 (session, NULL, AV3212D_ATTR_
          CONFIGURE_FREQUENCY,
          3212D_VAL_CONFIGURE_FREQUENCY_CH2);
   status = 3212D_SetAttributeViInt32 (session, NULL,
          AV3212D_ATTR_CONFIGURE_MEASURE_MODE,
          3212D_VAL_CONFIGURE_MEASURE_MODE_CW);
   status = 3212D_SetAttributeViInt32 (session, NULL,
          AV3212D_ATTR_CONFIGURE_MEASURE_TYPE,
          3212D_VAL_CONFIGURE_MEASURE_TYPE_CRF);
   ViStatus AV3212D_GetAttributeViReal64 (session,NULL,
          AV3212D_ATTR_ FETCH_CARRIER_FREQUENCY, &average);

   printf("CRF value is：%lf", average);
   // Close the session
   status = 3212D_close(session);
   // Close the session
   status = 3212D_close(session);
}
```

附录 SCPI 命令速查表

命令名称	命令功能
通用命令	
*CLS	清除状态字节寄存器
*ESE(?)	设置(查询)标准事件状态使能寄存器
*ESR?	查询标准事件状态寄存器(只能查询)
*IDN?	查询一个识别字符串
*OPC(?)	将标准事件状态寄存器第 0 位设为 0 （查询时始终返回为 1）
*RST	置为厂家预先定义的已知状态(只能设置)
*SRE(?)	设置(查询)服务请求使能寄存器
*STB(?)	查询状态字节(只能查询)
*TST?	执行自测试，并且返回自测试结果(只能查询)
*WAI	要求设备在当前命令完成前不执行其它命令(无查询)
*TRG	执行群触发命令(无查询)
标准命令	
:SYSTem:ERRor[NEXT]?	查询错误队列中的当前错误信息
:SYSTem:VERSion?	查询仪器遵循的 SCPI 规范的版本号，返回结果如：1999.0
:STATus:QUEStionable:CONDition?	这个查询返回问题条件数据寄存器的位值的和
:STATus:QUEStionable:PTRansition	该命令决定当位正向变化时（0 到 1），问题条件数据寄存器中的哪些位将设置问题事件数据寄存中对应得位
:STATus:QUEStionable:NTRansition	该命令决定当位负向变化时（1 到 0），问题条件数据寄存器中的哪些位将设置问题事件数据寄存中对应得位
:STATus:QUEStionable[:EVENt]	该查询返回问题事件数据寄存器的位值的和
:STATus:QUEStionable:ENABle	这个命令决定问题时间数据寄存器的哪些位将设置状态字节寄存器的问题状态组的概要位（bit 3）
:STATus:OPERation:CONDition?	该查询返回一个 10 进制的数，然后被设置为标准操作状态组的一部分
:STATus:OPERation:PTRansition	该命令决定当位正向变化时（0 到 1），标准操作条件寄存器中的哪些位将设置标准操作事件寄存中对应得位
:STATus:OPERation:NTRansition	该命令决定当位负向变化时（1 到 0），标准操作条件寄存器中的哪些位将设置标准操作事件寄存中对应得位
:STATus:OPERation[:EVENT]?	从标准操作事件寄存器返回一个 10 进制的数据
:STATus:OPERation:ENABle	该命令决定标准操作事件寄存器中的哪一位将设置在状态字节寄存器中的标 准操作状态概要位

:STATus:PRESet	该命令复位所有的传输滤波器、使能寄存器和错误/事件队列使能寄存器
测量命令集	
:CONFigure:MEASure:MODE CW PULse	设置测量模式：连续波 或 脉冲
:CONFigure:MEASure:MODE?	查询测量模式，返回值：CW，PUL
:CONFigure:MEASure:PULSe:MODE NORM|NPUL	设置脉冲测量模式：正常 或 窄脉冲
:CONFigure:MEASure:PULSe:MODE?	查询脉冲测量模式，返回值：NORM，NPUL
:CONFigure:MEASure:TYPE CRF	设置测量参数类型：载波频率
:FETCh:CARRier:FREQuency?	查询当前载波频率值，单位默认为 Hz
:CONFigure:MEASure:TYPE CRI	设置测量参数类型：载波周期
:FETCh:CARRier:PERiod?	查询当前载波周期值，单位默认为 ns
:CONFigure:MEASure:TYPE PWID	设置测量参数类型：脉冲宽度
:FETCh:PULSe:WIDth?	查询当前脉冲宽度值，单位默认为 ns
:CONFigure:MEASure:TYPE PRI;	设置测量参数类型：脉冲周期
:FETCh:PULSe:PERiod?	查询当前脉冲周期值，单位默认为 ns
:CONFigure:MEASure:TYPE PRF;	设置测量参数类型：脉冲频率
:FETCh:PULSe:FREQuency?	查询当前脉冲频率值，单位默认为 Hz
:CONFigure:MEASure:TYPE OFFT;	设置测量参数类型：脉冲关断
:FETCh:PULSe:OFF?	查询当前脉冲关断值，单位默认为 ns
:CONFigure:MEASure:TYPE DUTY;	设置测量参数类型：脉冲占空比
:CONFigure:MEASure:REDO	启动重新测量
:FETCh:PULSe:DUTY?	查询当前脉冲占空比值，单位默认为%
输入命令集	
:CONFigure:FREQuency (@1)|(@2)	设置测量通道：通道 1 | 通道 2
:CONFigure:FREQuency?	查询测量通道，返回值：(@1)，(@2)
:INPut1:IMPedance 50OHM|1MOHM	设置通道 1 阻抗：50Ω 或 1MΩ
:INPut1:IMPedance?	查询通道 1 阻抗，返回值：50OHM，1MOHM
手动命令集	
:CONFigure:CARRier:FREQuency:STATe OFF|ON	设置手动载波开关：关 或 开
:CONFigure:CARRier:FREQuency:STATe?	查询手动载波状态
:CONFigure:CARRier:FREQuency <numeric_value><unit>	设置手动载波频率。
:CONFigure:CARRier:FREQuency?	查询手动载波频率
:CONFigure:IF:STATe OFF|ON	设置手动中频开关：关 或 开
:CONFigure:IF:STATe?	查询手动中频状态
:CONFigure:IF <numeric_value><unit>	设置手动中频频率

:CONFigure:IF?	查询手动中频频率
自动命令集	
:CONFigure:AUTO:STAT OFF| ON	设置手动载波开关：关 或 开
:CONFigure:AUTO:STAT?	查询手动载波状态，返回值：0，1
:CONFigure:AUTO:BAND ALL|B0|B1|B2|B3|USER	设置全波段|0.5—0.8GHz|0.8—1.4GHz|1.4—2.6GHz|2.6—5GHz|5—9.8GHz|9.8—13.5GHz|13.5—20GHz|20—26.5GHz|26.5GHz—40GHz|40GHz—50GHz|USER
:CONFigure:AUTO:BAND?	查询波段
:CONFigure:USER:FREQ:STARt <numeric_value><unit>;	设置用户定义波段起始频率
:CONFigure:USER:FREQ:STARt?;	查询用户定义波段起始频率
:CONFigure:USER:FREQ:STOP <numeric_value><unit>;	设置用户定义波段终止频率
:CONFigure:USER:FREQ:STOP?	查询用户定义波段终止频率
分辨率命令集	
[:SENSe]:FREQuency:RESolution 0.1Hz|1Hz|10Hz|100Hz|1kHz|10kHz|100kHz|1MHz	设置分辨率：0.1Hz、1Hz、10Hz、100Hz、1kHz、10kHz、100kHz、1MHz
[:SENSe]:FREQuency:RESolution?	查询分辨率
[:SENSe]:AVERage:COUNt <numeric_value>	设置平均次数
[:SENSe]:AVERage:COUNt?	查询平均次数
[:SENSe]:SMOOth:STATe OFF|ON	设置平滑开关，关 或 开，默认关
[:SENSe]:SMOOth:STATe?	查询平滑开关
运算命令集	
:CONFigure:MATH:CRF:SCALe <numeric_value>	设置载波频率比例
:CONFigure:MATH:CRF:SCALe?	查询载波频率比例
:CONFigure:MATH:CRF:OFFSet <numeric_value><unit>	设置载波频率偏置
:CONFigure:MATH:CRF:OFFSet?	查询载波频率偏置
:CONFigure:MATH:CRI:SCALe <numeric_value>	设置载波周期比例
:CONFigure:MATH:CRI:SCALe?	查询载波周期比例
:CONFigure:MATH:CRI:OFFSet <numeric_value><unit>	设置载波周期偏置
:CONFigure:MATH:CRI:OFFSet?	查询载波周期偏置
:CONFigure:MATH:PRF:SCALe <numeric_value>	设置脉冲频率比例
:CONFigure:MATH:PRF:SCALe?	查询脉冲频率比例
:CONFigure:MATH:PRF:OFFSet <numeric_value><unit>	设置脉冲频率偏置
:CONFigure:MATH:PRF:OFFSet?	查询脉冲频率偏置
:CONFigure:MATH:PRI:SCALe <numeric_value>	设置脉冲周期比例
:CONFigure:MATH:PRI:SCALe?	查询脉冲周期比例

:CONFigure:MATH:PRI:OFFSet <numeric_value><unit>	设置脉冲周期偏置
:CONFigure:MATH:PRI:OFFSet?	查询脉冲周期偏置
:CONFigure:MATH:PWID:SCALe <numeric_value>	设置脉冲宽度比例
:CONFigure:MATH:PWID:SCALe?	查询脉冲宽度比例
:CONFigure:MATH:PWID:OFFSet <numeric_value><unit>	设置脉冲宽度偏置,
:CONFigure:MATH:PWID:OFFSet?	查询脉冲宽度偏置
:CONFigure:MATH:OFFT:SCALe <val>	设置脉冲关断比例
:CONFigure:MATH:OFFT:SCALe?	查询脉冲关断偏置
:CONFigure:MATH:OFFT:OFFSet <numeric_value><unit>	设置脉冲关断偏置
:CONFigure:MATH:OFFT:OFFSet?	查询脉冲关断偏置
:CONFigure:MATH:DUTY:SCALe <numeric_value>	设置脉冲占空比比例
:CONFigure:MATH:DUTY:SCALe?	查询脉冲占空比比例
:CONFigure:MATH:DUTY:OFFSet <numeric_value><unit>	设置脉冲占空比偏置
:CONFigure:MATH:DUTY:OFFSet?	查询脉冲占空比偏置
系统命令集	
:TRIGger:MODE INT|EXT|ARM	设置触发方式：内部门模式、外部门模式、ARM 模式
:TRIGger:MODE?	查询触发方式
:TRIGger:DELAy <val><unit>	设置触发延时
:TRIGger:DELAy?	查询触发延时
:SYSTem:SCOPe:STATe OFF|ON	设置 ScopeView 开关
:SYSTem:SCOPe:STATe?	查询 ScopeView
:CONFigure:DISPlay:LANGuage CN|EN	设置显示语言：中文 或 英文
:CONFigure:DISPlay:LANGuage?	查询语言
:SYSTem:COMMunicate:GPIB:ADDRess <val>	设置 GPIB 地址
:SYSTem:COMMunicate:GPIB:ADDRess?	查询 GPIB 地址
:SYSTem:COMMunicate:LAN:IP <val>	设置 IP 地址
:SYSTem:COMMunicate:LAN:IP?	查询 IP 地址
:SYSTem:COMMunicate:LAN:MASK <val>	设置子网掩码
:SYSTem:COMMunicate:LAN:MASK?	查询子网掩码
:SYSTem:COMMunicate:LAN:GATE <val>	设置网关
:SYSTem:COMMunicate:LAN:GATE?	查询网关
:SYSTem:COMMunicate:LAN:DNS <val>	设置 DNS
:SYSTem:COMMunicate:LAN:DNS?	查询 DNS
:CALibration:INSertion:STARt;	启动内插校准

:CALibration:SAVE	存储校准结果
:SYSTem:PRESet:TYPE NORMal USER	复位模式 系统 或 用户
:SYSTem:PRESet:TYPE	查询复位模式
:SYSTem:PRESet[:USER]:SAVE	存储用户复位设置
:SYSTem:SECurity:KB OFF| 0	关闭键盘，只可设置不可查询
:SYSTem:SECurity:DISPlay OFF| 0	关闭显示，只可设置不可查询