RIGOL

# Bluetooth Test

## For RSA6000 Series Spectrum Analyzer

Programming Guide

Apr. 2026

# Guaranty and Declaration



**Trademark Information**
RIGOL® is the trademark of RIGOL TECHNOLOGIES CO., LTD.

**Notices**
- **RIGOL** products are covered by P.R.C. and foreign patents, issued and pending.
- **RIGOL** reserves the right to modify or change parts of or all the specifications and pricing policies at the company's sole decision.
- Information in this publication replaces all previously released materials.
- Information in this publication is subject to change without notice.
- **RIGOL** shall not be liable for either incidental or consequential losses in connection with the furnishing, use, or performance of this manual, as well as any information contained.
- Any part of this document is forbidden to be copied, photocopied, or rearranged without prior written approval of **RIGOL**.

**Product Certification**
**RIGOL** guarantees that this product conforms to the national and industrial standards in China as well as the ISO9001:2015 standard and the ISO14001:2015 standard. Others international standard conformance certifications are in progress

**Contact Us**
If you have any problem or requirement when using our products or this manual, please contact **RIGOL**.
E-mail: service@rigol.com
Website: *http://www.rigol.com*

# Chapter 1 Document Overview

This manual is your guide to programming RSA6000 series spectrum analyzer to access the Bluetooth function by using SCPI commands through remote interface.
RSA6000 series can communicate with the PC via the USB or LAN (requiring to work with RIGOL USB-GPIB adaptor) interface.

**Tip**

For the latest version of this manual, download it from the official website of RIGOL (*http://www.rigol.com*).

**Publication Number**
PGD33100-1110

**Software Version**
00.01.00.01
Software upgrade might change or add product features. Please acquire the latest version of the manual from RIGOL website or contact RIGOL to upgrade the software.

**Format Conventions in this Manual**

**1. Key**

The front panel key is denoted by the menu key icon. For example, 

indicates the "System" key.

**2. Menu**

The menu item is denoted by the format of "Menu Name (Bold) + Character Shading" in the manual. For example, **Setup** indicates clicking or tapping the "Setup" sub-menu under the **System** menu to view the configuration items.

**3. Operation Procedures**

The next step of the operation is denoted by an arrow ">" in the manual. For example,  > **Save** indicates that first clicking or tapping the icon  , then clicking or tapping **Save**.

**4. Connector**

The connectors on the front or rear panel are usually denoted by the format of "Connector Name (Bold) + Square Brackets (Bold)". For example, **[TRIG IN]**.

**Content Conventions in this Manual**

The RSA6000 series spectrum analyzer includes the following models. Unless otherwise specified, this manual takes RSA6265 as an example to illustrate the functions and operation methods of RSA5065 series spectrum analyzer.

| Model | Frequency Range |
|---|---|
| RSA6265 | 5 kHz to 26.5 GHz |
| RSA6140 | 5 kHz to 14 GHz |
| RSA6085 | 5 kHz to 8.5 GHz |

# Contents

# Chapter 2Command System

This chapter introduces the Bluetooth commands of the RSA6000 series spectrum analyzer in VSA mode.

**Remarks:**

1. For the command set, unless otherwise specified, the query command returns "N/A" (without quotations in its return format) if no specified option is installed. If the queried function is disabled or improper type match is found, the query command will return "Error" (without quotations in its return format).
2. This manual takes RSA6265 as an example to illustrate the range of the parameters in each command.

# :CALCulate Commands

## :CALCulate:DDEMod:MARKer<n>:TRACe

**Syntax**
:CALCulate:DDEMod:MARKer<n>:TRACe <integer>
:CALCulate:DDEMod:MARKer<n>:TRACe?

**Description**
Sets the trace marked by the specified marker.
Queries the trace marked by the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <integer> | Discrete | 1\|2\|3 | 1 |

**Remarks**
When <integer> is set to 1, it indicates the marker trace is RF Envelope; when <integer> is set to 2, it indicates the marker trace is Demod Waveform; when <integer> is set to 3, it indicates the marker trace is Spectrum.

**Return Format**
The query returns an integer ranging from 1 to 3.

**Example**
The following command sets the marker trace marked by Marker 1 to Demod Waveform.
:CALCulate:DDEMod:MARKer1:TRACe 2

The following query returns 2.
:CALCulate:DDEMod:MARKer1:TRACe?

## :CALCulate:DDEMod:MARKer<n>:X

**Syntax**
:CALCulate:DDEMod:MARKer<n>:X <real>
:CALCulate:DDEMod:MARKer<n>:X?

**Description**
Sets the X-axis value of the specified marker.
Queries the X-axis value of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <real> | Real | Refer to "**Remarks**" | —— |

**Remarks**
<real> can be any value within the available range of the current X axis. In the RF Envelope and Demod Waveform view, X-axis represents time in s, and in the Spectrum view, X-axis represents frequency in Hz.
If the marker mode is set to Position or Fixed, this parameter is used to set the X value at the marker.
If the marker mode is set to Delta, this parameter is used to set the X value of the Delta marker in relative to that of the reference marker.

**Return Format**
The query returns the X-axis value for the specified marker in real number.

**Example**
The following command sets the X-axis value of Marker 1 to 2.402 GHz.
:CALCulate:DDEMod:MARKer1:X 2402000000

The following query returns 2402000000.
:CALCulate:DDEMod:MARKer1:X?

## :CALCulate:DDEMod:MARKer<n>:Y:REAL

**Syntax**
:CALCulate:DDEMod:MARKer<n>:Y:REAL <real>
:CALCulate:DDEMod:MARKer<n>:Y:REAL?

**Description**
Sets the Y value of the specified marker.
Queries the Y value of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <real> | Real | —— | -- |

**Remarks**
This command is only valid when the marker mode is set to Fixed. Before executing this command, select the marker trace view first.

**Return Format**
The query returns the Y-axis value for the marker in real number.

**Example**
The following command sets the Y-axis value of Marker 1 in Spectrum view to 10 dBm.
:CALCulate:DDEMod:MARKer1:Y:REAL 10

The following query returns 10.
:CALCulate:DDEMod:MARKer1:Y:REAL?

## :CALCulate:DDEMod:MARKer:SELect

**Syntax**
:CALCulate:DDEMod:MARKer:SELect <num>
:CALCulate:DDEMod:MARKer:SELect?

**Description**
Sets the selected marker.
Queries the currently selected marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <num> | Integer | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Return Format**
The query returns the current marker in integer.

**Example**
The following command sets Marker 2 to be the selected marker.
:CALCulate:DDEMod:MARKer:SELect 2

The following query command returns 2.
:CALCulate:DDEMod:MARKer:SELect?

## :CALCulate:DDEMod:MARKer:NEXT

**Syntax**
:CALCulate:DDEMod:MARKer:NEXT

**Description**
Switches to the next marker in the current view.

## :CALCulate:DDEMod:MARKer:COUPle[:STATe]

**Syntax**
:CALCulate:DDEMod:MARKer:COUPle[:STATe] <bool>
:CALCulate:DDEMod:MARKer:COUPle[:STATe]?

**Description**
Enables or disables the couple marker function.
Queries the on/off status of the couple marker function.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Remarks**
When you enable the couple marker function, moving any marker will make other markers (that are not fixed or off) move along with it.

**Return Format**
The query returns 0 or 1.

**Example**
The following command disables the couple marker function.
:CALCulate:DDEMod:MARKer:COUPle:STATe OFF or :CALCulate:DDEMod:MARKer:COUPle:STATe 0

The following query returns 0.
:CALCulate:DDEMod:MARKer:COUPle:STATe?

## :CALCulate:DDEMod:MARKer:AOFF

**Syntax**
:CALCulate:DDEMod:MARKer:AOFF

**Description**
Turns off all the enabled markers.

## :CALCulate:DDEMod:MARKer<n>:MODE

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MODE <mode>
:CALCulate:DDEMod:MARKer<n>:MODE?

**Description**

Sets the type of the specified marker.
Queries the type of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <mode> | Discrete | POSition\|DELTa\|FIXed\|OFF | POSition |

**Remarks**
POSition: indicates the normal marker.
DELTa: indicates difference between two data points.
FIXed: indicates that the marker is fixed.
OFF: turns off the selected marker.

**Return Format**
The query returns POS, DELT, FIX, or OFF.

**Example**
The following command sets the type of Marker 1 to Position.
:CALCulate:DDEMod:MARKer1:MODE POSition

The following query returns POS.
:CALCulate:DDEMod:MARKer1:MODE?

## :CALCulate:DDEMod:MARKer<n>[:SET]:CENTer

**Syntax**
:CALCulate:DDEMod:MARKer<n>[:SET] CENTer

**Description**
Sets the frequency of the specified marker to center frequency of the analyzer.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum.
If the marker mode of the specified marker is Position or Fixed, the center frequency will be set to the frequency of the marker.
If the specified marker mode is Delta, the center frequency will be set to the frequency of the Delta marker.

**Example**
The following command sets the frequency of Marker 1 (Position) to the center frequency of the analyzer.
:CALCulate:DDEMod:MARKer1:SET:CENTer

## :CALCulate:DDEMod:MARKer<n>[:SET]:DELTa:CENTer

**Syntax**
:CALCulate:DDEMod:MARKer<n>[:SET]:DELTa:CENTer

**Description**
Sets the center frequency of the analyzer to the frequency difference between the two Delta markers.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum and the marker mode is set to Delta.

Example
The following command sets the center frequency of the analyzer to the frequency difference of the Delta marker 1.
:CALCulate:DDEMod:MARKer1:SET:DELTa:CENTer

## :CALCulate:DDEMod:MARKer<n>[:SET]:RLEVel

**Syntax**
:CALCulate:DDEMod:MARKer<n>[:SET]:RLEVel

**Description**
Sets the reference level of the analyzer to the amplitude of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
If the marker mode of the specified marker is Position or Fixed, the reference level will be set to the amplitude of the marker.
If the specified marker mode is Delta and the current marker is the reference marker, then the reference level is set to the amplitude of the reference marker; if the current marker is the Delta marker, then the reference level is set to the amplitude of the Delta marker.

**Example**
The following command sets the reference level of the analyzer to the amplitude of Marker 2 (Position).
:CALCulate:DDEMod:MARKer2:SET:RLEVel

## :CALCulate:DDEMod:MARKer<n>[:SET]:STEP

**Syntax**
:CALCulate:DDEMod:MARKer<n>[:SET]:STEP

**Description**
Sets the CF step of the analyzer to the frequency of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum.
If the marker mode of the specified marker is Position or Fixed, CF Step will be set to the Freq of the marker.
If the specified marker mode is Delta, the CF step will be set to the frequency of the Delta marker.

**Example**
The following command sets the CF Step of the analyzer to the Freq of Marker 4 (Position).
:CALCulate:DDEMod:MARKer4:SET:STEP

## :CALCulate:DDEMod:MARKer<n>:REFerence

**Syntax**
:CALCulate:DDEMod:MARKer<n>:REFerence <integer>
:CALCulate:DDEMod:MARKer<n>:REFerence?

**Description**
Sets the reference marker for the specified marker.
Queries the reference marker for the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <integer> | Integer | 1 to 8 | By default, the reference marker is the marker next to it. |

**Remarks**
Each marker can have another marker to be its reference marker.
It the current marker is a Delta marker, the measurement result of the marker will be determined by the reference marker.
Any marker cannot have itself to be the reference marker.

**Example**
The following command sets the reference marker for Marker 1 to 2.
:CALCulate:DDEMod:MARKer1:REFerence 2

The following query returns 2.
:CALCulate:DDEMod:MARKer1:REFerence?

## :CALCulate:DDEMod:MARKer<n>:CPSearch[:STATe]

**Syntax**
:CALCulate:DDEMod:MARKer<n>:CPSearch[:STATe] <bool>
:CALCulate:DDEMod:MARKer<n>:CPSearch[:STATe]?

**Description**
Sets the on/off status of continuous peak search.
Queries the on/off status of continuous peak search.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Remarks**
N/A

**Example**
The following command enables the continuous peak search of Marker 1.
:CALCulate:DDEMod:MARKer1:CPSearch:STATe ON or :CALCulate:DDEMod:MARKer1:CPSearch:STATe 1

The following query returns 1.
:CALCulate:DDEMod:MARKer1:CPSearch:STATe?

## :CALCulate:DDEMod:MARKer<n>:MAXimum

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MAXimum

**Description**
Performs one peak search and marks it with the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
When no peak is found, a prompt message "No peak found" is displayed on the screen.

**Example**
The following command performs one peak search, and marks with Marker 2.
:CALCulate:DDEMod:MARKer2:MAXimum

## :CALCulate:DDEMod:MARKer<n>:MAXimum:LEFT

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MAXimum:LEFT

**Description**
Searches for and marks the peak closest to the left side of the current peak.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
When no peak is found, a prompt message "No peak found" is displayed on the screen.

**Example**
The following command performs one left peak search, and marks with Marker 2.
:CALCulate:DDEMod:MARKer2:MAXimum:LEFT

## :CALCulate:DDEMod:MARKer<n>:MAXimum:NEXT

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MAXimum:NEXT

**Description**
Searches for and marks the peak whose amplitude on the trace is next lower than that of the current peak.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
When no peak is found, a prompt message "No peak found" is displayed on the screen.

**Example**
The following command performs one next peak search, and marks with Marker 2.
:CALCulate:DDEMod:MARKer2:MAXimum:NEXT

## :CALCulate:DDEMod:MARKer<n>:MAXimum:RIGHt

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MAXimum:RIGHt

**Description**
Searches for and marks the peak closest to the right side of the current peak.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
When no peak is found, a prompt message "No peak found" is displayed on the screen.

**Example**
The following command performs one right peak search, and marks with Marker 2.
:CALCulate:DDEMod:MARKer2:MAXimum:RIGHt

## :CALCulate:DDEMod:MARKer<n>:MINimum

**Syntax**
:CALCulate:DDEMod:MARKer<n>:MINimum

**Description**
Searches for and marks the peak with the minimum amplitude on the trace.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | —— |

**Remarks**
When no peak is found, a prompt message "No peak found" is displayed on the screen.

**Example**
The following command performs one minimum search, and marks it with Marker 2.
:CALCulate:DDEMod:MARKer2:MINimum

## :CALCulate:MARKer<n>:PTPeak

**Syntax**
:CALCulate:MARKer<n>:PTPeak

**Description**
Performs the Pk-Pk search.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | 1 |

**Remarks**
After the command is executed, the marker mode of the specified marker selects "Delta" automatically. The peak search results will be marked by the reference marker (by default, the next marker), and the minimum search will be marked by the Delta marker.

**Example**

The following command performs the peak-peak search, and marks the peak-peak position with the reference marker (Marker 2) and the Delta marker (Marker 1), respectively.
:CALCulate:MARKer1:PTPeak

## :CALCulate:MARKer<n>[:SET]:DELTa:SPAN

**Syntax**
:CALCulate:MARKer<n>[:SET]:DELTa:SPAN

**Description**
Sets the frequency difference of the specified Delta marker to the span of the analyzer.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | 1 |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum and the marker mode is set to Delta.

**Example**
The following command sets the frequency difference of the Delta Marker 1 to the span of the analyzer.
:CALCulate:MARKer1:SET:DELTa:SPAN

## :CALCulate:MARKer<n>[:SET]:STARt

**Syntax**
:CALCulate:MARKer<n>[:SET]:STARt

**Description**
Sets the frequency of the specified marker to the start frequency of the analyzer.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | 1 |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum.
If the marker mode of the specified marker is Position or Fixed, the start frequency will be set to the frequency of the marker.
If the specified marker mode is Delta, the start frequency will be set to the frequency of the Delta marker.

**Example**
The following command sets the frequency of Marker 3 (Position) to the start frequency of the analyzer.
:CALCulate:MARKer3:SET:STARt

## :CALCulate:MARKer<n>[:SET]:STOP

**Syntax**
:CALCulate:MARKer<n>[:SET]:STOP

**Description**
Sets the frequency of the specified marker to the stop frequency of the analyzer.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | 1 |

**Remarks**
This command only takes effect when the marker trace is set to Spectrum.
If the marker mode of the specified marker is Position or Fixed, the stop frequency will be set to the frequency of the marker.
If the specified marker mode is Delta, the stop frequency will be set to the frequency of the Delta marker.

**Example**
The following command sets the frequency of Marker 2 (Position) to the stop frequency of the analyzer.
:CALCulate:MARKer2:SET:STOP

## :CALCulate:MARKer<n>:TRACe:AUTO

**Syntax**
:CALCulate:MARKer<n>:TRACe:AUTO <bool>
:CALCulate:MARKer<n>:TRACe:AUTO?

**Description**
Enables or disables the auto trace marking of the specified marker.
Queries the status of the auto trace marking of the specified marker.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4\|5\|6\|7\|8 | 1 |
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Remarks**
When you disable the auto marking of the trace, the currently enabled marker will stay on the corresponding trace.

**Return Format**
The query returns 0 or 1.

**Example**
The following command sets the marker trace of Marker 1 to be Auto.
:CALCulate:MARKer1:TRACe:AUTO ON or :CALCulate:MARKer1:TRACe:AUTO 1

The following query returns 1.
:CALCulate:MARKer1:TRACe:AUTO?

# :CALibration Commands

## :CALibration[:ALL]

**Syntax**
:CALibration[:ALL]

**Description**
Executes self-calibration immediately.

**Example**
The following command executes the self-calibration immediately.
:CALibration:ALL

## :CALibration:AUTO

**Syntax**
:CALibration:AUTO <bool>
:CALibration:AUTO?

**Description**
Enables or disables auto calibration.
Queries the setting status of auto calibration.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
The following command enables auto calibration.
:CALibration:AUTO ON or :CALibration:AUTO 1

The following query returns 1.
:CALibration:AUTO?

# :CONFigure Commands

## :CONFigure?

**Syntax**
:CONFigure?

**Description**
Queries the current measurement function.

**Return Format**
The query returns BLE_TX.

## :CONFigure:CATalog?

**Syntax**
:CONFigure:CATalog?

**Description**
Queries the currently supported measurement application.

**Return Format**
The query returns the measurement application names of the current working mode in strings, separated by commas.

# :DISPlay Commands

## :DISPlay:WINDow[<n>]:SWITch

**Syntax**
:DISPlay:WINDow[<n>]:SWITch <integer>

**Description**
Switches the view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3\|4 | —— |
| <integer> | Discrete | Refer to "Remarks" | —— |

**Remarks**
When <n> is set to 1, it indicates the RF Envelope view.
When <n> is set to 2, it indicates the Demod Waveform view.
When <n> is set to 3, it indicates the Spectrum view.
When <n> is set to 4, it indicates the Metrics view.

**Example**
The following command switches the view from 1 to 3.
:DISPlay:WINDow1:SWITch 3

## :DISPlay:DDEMod:WINDow[<n>]:Y[:SCALe]:AUTO:ONCE

**Syntax**
:DISPlay:DDEMod:WINDOW<n>:Y[:SCALe]:AUTO:ONCE

**Description**
Performs the Y-axis auto scaling in the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3 | 1 |

**Remarks**
When <n> is set to 1, it indicates the RF Envelope view.
When <n> is set to 2, it indicates the Demod Waveform view.
When <n> is set to 3, it indicates the Spectrum view.

**Example**
The following command performs the Y-axis auto scaling in the Demod Waveform view.
:DISPlay:DDEMod:WINDow2:Y:SCALe:AUTO:ONCE

## :DISPlay:TX:VIEW:ALL

**Syntax**
:DISPlay:TX:VIEW:ALL <num>
:DISPlay:TX:VIEW:ALL?

**Description**
Sets the combination view for the BTX measurement.
Queries the combination view for the BTX measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <num> | Integer | 2 to 30 | 30 |

**Remarks**
RF Envelope view: indicated by 2.
Demod Waveform view: indicated by 4.
Spectrum view: indicated by 8.
Metrics view: indicated by 16.
The value of the <num> parameter is the sum of the specified view number for all the displayed views.

**Return Format**
The query returns an integer ranging from 2 to 30.

**Example**
The following command sets the combination view to Spectrum and RF Envelope.
:DISPlay:TX:VIEW:ALL 10

The following query returns 10.
:DISPlay:TX:VIEW:ALL?

## :DISPlay:TX:VIEW:SELect

**Syntax**
:DISPlay:TX:VIEW:SELect <enum>
:DISPlay:TX:VIEW:SELect?

**Description**
Selects the combination view for the BTX measurement.
Queries the combination view for the BTX measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | QUAD\|RFSpectrum\|RFENvelope\|DWAVeform\|NRESults | QUAD |

**Remarks**
QUAD: displays all the views for the BTX measurement.
RFSpectrum: displays the Spectrum view and Metrics view.
DWAVeform: displays the Demod Waveform view and Metrics view.
RFENEnvelope: displays the RF Envelope view and Metrics view.
NRESults: displays Metrics view.

**Return Format**
The query returns QUAD, RFSP, RFEN, DWAV, or NRES.

**Example**
The following command sets the preset combination view to RFSpectrum.
:DISPlay:TX:VIEW:SELect RFSpectrum

The following query returns RFSP.
:DISPlay:TX:VIEW:SELect?

## :DISPlay:TX:VIEW:ENABle

**Syntax**
:DISPlay:TX:VIEW:ENABle <enum>,<bool>
:DISPlay:TX:VIEW:ENABle? <enum>

**Description**
Enables or disables the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | 1\|2\|3\|4 | —— |
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Remarks**
<enum> = 1: RF Envelope view
<enum> = 2: Demod Waveform view
<enum> = 3: Spectrum view
<enum> = 4: Metrics view

**Return Format**
The query returns 1 or 0.

**Example**
The following command enables the Spectrum view.
:DISPlay:TX:VIEW:ENABle 3,ON or :DISPlay:TX:VIEW:ENABle 3,1
The following query returns 1.
:DISPlay:TX:VIEW:ENABle? 3

## :DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:RLEVel

**Syntax**
:DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:RLEVel <real>
:DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:RLEVel?

**Description**
Sets the X-axis reference value of the specified view.
Queries the X-axis reference value of the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2 | 1 |
| <real> | Real | 0 to acq time – (X Scale/Div x 10) | 0 |

**Remarks**
<n> = 1: RF Envelope view; <n> = 2: Demod Waveform view.
After running this command, the X Scale/Div value will be recalculated.

**Return Format**
The query returns the X Ref Value of the specified view in real number. The unit is s.

**Example**
The following command sets the X-axis reference value of RF Envelope view to 1 ms.
:DISPlay:TX:WINDow1:TRACe:X:SCALe:RLEVel 0.001

The following query returns 0.001.
:DISPlay:TX:WINDow2:TRACe:X:SCALe:RLEVel?

## :DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:PDIVision

**Syntax**
:DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:PDIVision <rel>

:DISPlay:TX:WINDow[<n>]:TRACe:X[:SCALe]:PDIVision?

**Description**
Sets the X Scale/Div value for the specified view.
Queries the X Scale/Div value for the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2 | 1 |
| <rel> | Real | 0.1 µs to (Acq Time – X Ref Value)/10 | Refer to Remarks |

**Remarks**
When <n> is set to 1, it indicates the RF Envelope view. The default value of <rel> is 300 µs.
When <n> is set to 2, it indicates the Demod Waveform view. The default value of <rel> is 10 µs.

**Return Format**
The query returns the X Scale/Div value of the specified view in real number. The unit is s.

**Example**
The following command sets the X Scale/Div value to 1 µs.
:DISPlay:TX:WINDow1:TRACe:X:SCALe:PDIVision 0.000001

The following query returns 0.000001.
:DISPlay:TX:WINDow1:TRACe:X:SCALe:PDIVision?

## :DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:RLEVel

**Syntax**
:DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:RLEVel <real>
:DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:RLEVel?

**Description**
Sets the Y-axis reference value of the specified view.
Queries the Y-axis reference value of the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3 | 1 |
| <real> | Real | (-10 GHz + 10 Hz) to (10 GHz - 10 Hz) (Demod Waveform view)<br>-169.00 dBm to 30.00 dBm (Other views) | Refer to Remarks |

**Remarks**
<n> = 1: RF Envelope view; <n> = 2: Demod Waveform view; <n> = 3: Spectrum view
The default Y-axis reference value in the RF Envelope view is -1.2 dBm; the default Y-axis reference value in other views is 0.

**Return Format**
The query returns the Y-axis reference value in real number. The unit is determined by the selected Y-axis unit.

**Example**
The following command sets the Y-axis reference value of the RF Envelope view to 1 dBm.
:DISPlay:TX:WINDow1:TRACe:Y:SCALe:RLEVel 1

The following query returns 1.
:DISPlay:TX:WINDow1:TRACe:Y:SCALe:RLEVel?

## :DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:PDIVision

**Syntax**
:DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:PDIVision <rel>
:DISPlay:TX:WINDow[<n>]:TRACe:Y[:SCALe]:PDIVision?

**Description**
Sets the Y Scale/Div value for the specified view.
Queries the Y Scale/Div value for the specified view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Discrete | 1\|2\|3 | 1 |
| <rel> | Real | 1.00 Hz to 2 GHz (Demod Waveform view)<br>0.1 dBm to 20 dBm (Other views) | 50.00 kHz (Demod Waveform view)<br>10 dBm (Other views) |

**Remarks**
<n> = 1: RF Envelope view; <n> = 2: Demod Waveform view; <n> = 3: Spectrum view

**Return Format**
The query returns the Y Scale/Div value in real number. The unit is determined by the selected Y-axis unit.

**Example**
The following command sets the Y Scale/Div value to 100 Hz.
:DISPlay:TX:WINDow2:TRACe:Y:SCALe:PDIVision 100

The following query returns 100.
:DISPlay:TX:WINDow2:TRACe:Y:SCALe:PDIVision?

# :FETCh Commands

## :FETCh:DDEMod<n>?

**Syntax**
:FETCh:DDEMod<n>?

**Description**
Queries the digital demodulation measurement result.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <n> | Integer | 1\|2\|3\|4 | —— |

**Remarks**
When <n> is set to 1, the query returns the measurement trace data of the RF Envelope view.
When <n> is set to 2, the query returns the measurement trace data of the Demod Waveform view.
When <n> is set to 3, the query returns the measurement trace data of the Spectrum view.
When <n> is set to 4, the query returns the measurement trace data of the Metrics view.

**Return Format**
The query returns the measurement results in strings, separated by commas.

# IEEE 488.2 Common Commands

IEEE 488.2 common commands are used to operate or query the status registers.

## *CLS

**Syntax**
*CLS

**Description**
Clears all the event registers and status byte registers.

## *ESE

**Syntax**
*ESE <value>
*ESE?

**Description**
Sets the enable register for the standard event status register.
Queries the enable register for the standard event status register.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <value> | Integer | Refer to "**Remarks**" | 0 |

**Remarks**
The bit 2, bit 3, bit 4, and bit 7 are reserved; you can set their values but they will not affect the system. Bit 1 and bit 6 are not used and are always treated as 0; therefore, the range of <value> are the decimal numbers corresponding to the binary numbers ranging from 00000000 (0 in decimal) to 11111111 (255 in decimal) and of which bit 1 and bit 6 are 0.

**Return Format**
The query returns an integer. The integer equals to the binary-weighted sum of all the bits set in the register. For example, the query returns 144 if Bit 4 (16 in decimal) and Bit 7 (128 in decimal) are enabled.

**Example**
The following command sets the enable register for the standard event status register to 16.
*ESE 16

The following query returns 16.
*ESE?

## *ESR?

**Syntax**
*ESR?

**Description**
Queries and clears the event register for the standard event status register.

**Remarks**
Bit 1 and Bit 6 in the standard event status register are not in use, and are regarded as 0. The query returns a decimal value that corresponds to the binary values ranging from 00000000 (0 in decimal) to 11111111 (255 in decimal) and of which Bit 1 and Bit 6 are 0.

**Return Format**
The query returns an integer. The integer equals to the binary-weighted sum of all the bits set in the register. For example, the query returns 144 if Bit 4 (16 in decimal) and Bit 7 (128 in decimal) are enabled.

**Example**
The following query returns 24 (Bit 3 and Bit 4 have been set).
*ESR?

## *IDN?

**Syntax**
*IDN?

**Description**
Queries the ID string of instrument.

**Return Format**
The query returns the ID string in the following format:
RIGOL TECHNOLOGIES,<model>,<serial number>,XX.XX.XX
<model>: instrument model
<serial number>: serial number of the instrument
XX.XX.XX: firmware version with the release date

**Example**
The following query returns RIGOL TECHNOLOGIES,RSA6085,RSA60000000000,00.01.01.00.00@2026-03-09 15:01:12.
*IDN?

## *OPC

**Syntax**
*OPC
*OPC?

**Description**
Sets Bit 0 (Operation Complete, OPC) in the standard event status register to 1 after the current operation is finished.
Queries whether the current operation is finished.

**Return Format**
The query returns 1 after the current operation is finished; otherwise, the query returns 0.

## *RCL

**Syntax**
*RCL <integer>

**Description**
Recalls the selected register.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <integer> | Integer | 1 to 16 | —— |

**Example**
The following command recalls Register 1.

*RCL 1

## *RST

**Syntax**
*RST

**Description**
Restores the instrument to its factory default settings.

## *SAV

**Syntax**
*SAV <integer>

**Description**
Saves the current instrument state to the selected register.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <integer> | Integer | 1 to 16 | —— |

**Example**
The following command saves the current instrument state to Register 1.
*SAV 1

## *SRE

**Syntax**
*SRE <value>
*SRE?

**Description**
Sets the enable register for the status byte register.
Queries the bits in the service request register.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <value> | Integer | Refer to "**Remarks**" | 0 |

**Remarks**
Bit 0 and Bit 1 are not used and are always treated as 0; therefore, the range of <value> are the decimal numbers corresponding to the binary numbers ranging from 00000000 (0 in decimal) to 11111111 (255 in decimal) and of which Bit 0 and Bit 1 are 0.

**Return Format**
The query returns an integer. The integer equals to the binary-weighted sum of all the bits set in the register. For example, the query returns 144 if Bit 4 (16 in decimal) and Bit 7 (128 in decimal) are enabled.

**Example**
The following command sets the enable register for the status byte register to 16.
*SRE 16

The following query returns 16.
*SRE?

## *STB?

**Syntax**
*STB?

**Description**
Queries the event register for the status byte register.

**Remarks**
Bit 0 and Bit 1 in the status byte register are not in use, and are regarded as 0. The query returns a decimal value that corresponds to the binary values ranging from 00000000 (0 in decimal) to 11111111 (255 in decimal) and of which Bit 0 and Bit 1 are 0.

**Return Format**
The query returns an integer. The integer equals to the binary-weighted sum of all the bits set in the register. For example, the query returns 144 if Bit 4 (16 in decimal) and Bit 7 (128 in decimal) are enabled.

**Example**
The following query returns 24 (Bit 3 and Bit 4 have been set).
*STB?

## *TST?

**Syntax**
*TST?

**Description**
Queries whether the self-check operation is finished.

**Remarks**
The query returns 0 or 1. A zero is returned if the test is successful, 1 if it fails.

## *WAI

**Syntax**
*WAI

**Description**
Waits for all the pending operations to complete before executing any additional commands.

## :GPIB:PARSe:END

**Syntax**
:GPIB:PARSe:END

**Description**
The GPIB module command. This instruction set sends the "Parse End" command to the GPIB module. If this command is not sent, the USB-GPIB adapter module fails to work. You will receive no return value after sending a command.

# :INITiate Commands

## :INITiate:CONTinuous

**Syntax**
:INITiate:CONTinuous <bool>
:INITiate:CONTinuous?

**Description**
Selects continuous (ON|1) or single (OFF|0) measurement mode.
Queries the current measurement mode.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Return Format**
The query returns 0 or 1.

**Example**
The following command sets the instrument to sweep continuously.
:INITiate:CONTinuous ON or :INITiate:CONTinuous 1

The following query returns 1.
:INITiate:CONTinuous?

## :INITiate:RESTart

**Syntax**
:INITiate:RESTart

**Description**
Restarts to initiate a sweep.

# :INPut Commands

## :INPut:IF:OVERload?

**Syntax**
:INPut:IF:OVERload?

**Description**
Queries the IF overload.

**Remarks**
The query returns 0 or 1. 0 indicates not overloaded; 1 indicates overloaded.

# :INSTrument Commands

## :INSTrument:COUPle:FREQuency:CENTer

**Syntax**
:INSTrument:COUPle:FREQuency:CENTer <enum>
:INSTrument:COUPle:FREQuency:CENTer?

**Description**
Sets the setting status of the global center frequency of the instrument.
Queries the setting status of the global center frequency of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | ALL\|NONE | NONE |

**Remarks**
NONE: disables the global CF mode.
ALL: enables the global CF mode.
If you execute this command in any mode, the center frequency of the current mode is set to the global center frequency. Adjusting the center frequency in a mode, while the global center frequency is on, will modify the global center frequency.

**Return Format**
The query returns ALL or NONE.

**Example**
The following command enables the global center frequency of the instrument.
:INSTrument:COUPle:FREQuency:CENTer ALL

The following query returns ALL.
:INSTrument:COUPle:FREQuency:CENTer?

## :INSTrument:SCReen:CATalog?

**Syntax**
:INSTrument:SCReen:CATalog?

**Description**
Queries the available measurement modes displayed at the bottom of the screen.

## :INSTrument:SCReen:CREate

**Syntax**
:INSTrument:SCReen:CREate

**Description**
Creates a new measurement mode label at the bottom of the screen.

## :INSTrument:SCReen:DELete

**Syntax**
:INSTrument:SCReen:DELete

**Description**

Deletes the currently selected measurement mode label at the bottom of the screen.

## :INSTrument:SCReen:DELete:ALL

**Syntax**
:INSTrument:SCReen:DELete:ALL

**Description**
Deletes all the currently not selected measurement mode label at the bottom of the screen.

## :INSTrument:SCReen:REName

**Syntax**
:INSTrument:SCReen:REName <name>

**Description**
Renames the currently selected measurement mode label at the bottom of the screen.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <name> | ASCII String | The label can contain English letters and numbers, as well as some symbols. | —— |

## :INSTrument:SCReen:SELect

**Syntax**
:INSTrument:SCReen:SELect <name>
:INSTrument:SCReen:SELect?

**Description**
Switches to the specified screen.
Queries the currently selected screen.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <name> | ASCII String | The label can contain English letters and numbers, as well as some symbols. | —— |

**Remarks**
You can run the *:INSTrument:SCReen:CATalog?* command to query the measurement mode label name.

**Example**
The following command selects the measurement mode label VSA TX 1.
:INSTrument:SCReen:SELect VSA TX 1

The following query returns VSA TX 1.
:INSTrument:SCReen:SELect?

## :INSTrument[:SELect]

**Syntax**
:INSTrument[:SELect] <enum>
:INSTrument[:SELect]?

**Description**
Selects the working mode of the instrument.
Queries the working mode of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | SA\|GPSA_TG\|RTSA\|RTSA_UFS\|GPSA_ACP\|GPSA_CNR\|GPSA_CHPower\|GPSA_TOI\|GPSA_HARModist\|GPSA_OBW\|GPSA_TPOWer\|EMI\|VSA\|AM\|FM\|PM\|VSA_ZIGBEE\|VSA_LORA\|VSA_TX\|GPSA_IBEM\|PNOISE\|PULSE | VSA_TX |

**Remarks**
Only VSA_TX is supported.

**Example**
The following command sets the working mode of the instrument to VSA_TX.
:INSTrument:SELect VSA_TX

The following query returns VSA_TX.
:INSTrument:SELect?

# :LAN Commands

## :LAN:DHCP

**Syntax**
:LAN:DHCP <bool>
:LAN:DHCP?

**Description**
Enables or disables the DHCP configuration; queries the on/off status of DHCP configuration.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Remarks**
When the three IP configuration types (DHCP, Auto IP, and Static IP) are all turned on, the priority of the parameter configuration from high to low is "DHCP", "Auto IP", and "Static IP". The three IP configuration types cannot be all turned off at the same time.
In DHCP mode, the DHCP server on the current network assigns network parameters (such as the IP address) to the instrument.
After the :LAN:APPLy command is executed, the configuration type can take effect immediately.

**Return Format**
The query returns 0 or 1.

**Example**
:LAN:DHCP OFF /*Disables DHCP configuration.*/
:LAN:DHCP? /*The query returns 0.*/

## :LAN:AUToip

**Syntax**
:LAN:AUToip <bool>
:LAN:AUToip?

**Description**
Enables or disables the Auto IP configuration; queries the on/off status of Auto IP configuration.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Remarks**
When the auto IP mode is valid, disable DHCP manually. In this mode, the instrument acquires the IP address ranging from "169.254.0.1" to "169.254.255.254" and the subnet mask (255.255.0.0) automatically according to the current network configuration.

**Return Format**
The query returns 0 or 1.

**Example**
:LAN:AUToip OFF /*Disables Auto IP configuration.*/
:LAN:AUToip /*The query returns 0.*/

## :LAN:MANual

**Syntax**
:LAN:MANual <bool>
:LAN:MANual?

**Description**
Enables or disables the static IP configuration; queries the on/off status of static IP configuration.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Remarks**
When the static IP mode is valid, disable DHCP and Auto IP manually. You can self-define the network parameters of the instrument, such as IP address, subnet mask, gateway, and DNS address. For the setting of the IP address, refer to the :LAN:IPADdress command. For the setting of the subnet mask, refer to the :LAN:SMASk command. For the setting of the gateway, refer to the :LAN:GATeway command. For the setting of the DNS address, refer to the :LAN:DNS command.

**Return Format**
The query returns 0 or 1.

**Example**
:LAN:MANual ON /*Enables the static IP configuration.*/
:LAN:MANual? /*The query returns 1.*/

## :LAN:IPADdress

**Syntax**
:LAN:IPADdress <string>
:LAN:IPADdress?

**Description**
Sets or queries the instrument IP address.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <string> | ASCII String | Refer to "**Remarks**" | —— |

**Remarks**
The format of the<string> is "nnn.nnn.nnn.nnn". The range for the first segment (nnn) of the address is from 0 to 223 (except 127); and the range for the other three segments is from 0 to 255.
Only when the IP configuration mode is static IP and both DHCP and Auto IP are disabled, can you use the set the IP address.

**Return Format**
The query returns the current IP address in strings.

**Example**
:LAN:IPADdress 192.168.1.10 /*Sets the IP address to 192.168.1.10*/
:LAN:IPADdress? /*The query returns 192.168.1.10*/

## :LAN:SMASk

**Syntax**
:LAN:SMASk <string>
:LAN:SMASk?

**Description**
Sets or queries the subnet mask.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <string> | ASCII String | Refer to "**Remarks**" | —— |

**Remarks**
The format of <string> is "nnn.nnn.nnn.nnn". The range for the first segment (nnn) is from 0 to 255.
Only when the IP configuration mode is static IP and both DHCP and Auto IP are disabled, can you use the set subnet mask.

**Return Format**
The query returns the current subnet mask in strings.

**Example**
:LAN:SMASk 255.255.255.0 /*Sets the subnet mask to 255.255.255.0.*/
:LAN:SMASK? /*The query returns 255.255.255.0.*/

## :LAN:GATeway

**Syntax**
:LAN:GATeway <string>
:LAN:GATeway?

**Description**
Sets or queries the default gateway.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <string> | ASCII String | Refer to "**Remarks**" | —— |

**Remarks**
The format of the<string> is "nnn.nnn.nnn.nnn". The range for the first segment (nnn) of the address is from 0 to 223 (except 127); and the range for the other three segments is from 0 to 255.
Only when the IP configuration mode is static IP and both DHCP and Auto IP are disabled, can you use the set default gateway.

**Return Format**
The query returns the current gateway in strings.

**Example**
:LAN:GATeway 192.168.1.1 /*Sets the default gateway to 192.168.1.1.*/
:LAN:GATeway? /*The query returns 192.168.1.1.*/

## :LAN:DNS

**Syntax**
:LAN:DNS <string>
:LAN:DNS?

**Description**

Sets or queries the DNS address.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <string> | ASCII String | Refer to "**Remarks**" | —— |

**Remarks**
The format of the<string> is "nnn.nnn.nnn.nnn". The range for the first segment (nnn) of the address is from 0 to 223 (except 127); and the range for the other three segments is from 0 to 255.
Only when the IP configuration mode is static IP and both DHCP and Auto IP are disabled, can you use the set DNS.

**Return Format**
The query returns the current DNS address in strings.

**Example**
:LAN:DNS 192.168.1.1 /*Sets the DNS address to 192.168.1.1.*/
:LAN:DNS? /*The query returns 192.168.1.1.*/

## :LAN:MAC?

**Syntax**
:LAN:MAC?

**Description**
Queries the MAC address of the instrument.

**Return Format**
The query returns the MAC address in strings. For example, 00:19:AF:00:11:22.

## :LAN:DSERver?

**Syntax**
:LAN:DSERver?

**Description**
Queries the DHCP server address.

**Return Format**
The query returns the DHCP server address in strings.

## :LAN:STATus?

**Syntax**
:LAN:STATus?

**Description**
Queries the current network configuration status.

**Remarks**
- UNLINK: Not connected!
- CONNECTED: Connected successfully.
- INIT: acquiring IP address.
- IPCONFLICT: IP conflicted.
- BUSY: please wait...

- CONFIGURED: network configuration succeeded.
- DHCPFAILED: DHCP configuration failed.
- INVALIDIP: invalid IP.
- IPLOSE: IP lost.

**Return Format**
The query returns UNLINK, CONNECTED, INIT, IPCONFLICT, BUSY, CONFIGURED, DHCPFAILED, INVALIDIP, or IPLOSE.

## :LAN:VISA?

**Syntax**
:LAN:VISA? [<type>]

**Description**
Queries the VISA address of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <type> | Discrete | {USB\|LXI} | —— |

**Remarks**
This command contains a parameter "type" and it is used to set or query the address type. By default, it returns the LXI address.

**Return Format**
The query returns the VISA address in strings.

## :LAN:MDNS

**Syntax**
:LAN:MDNS <bool>
:LAN:MDNS?

**Description**
Enables or disables mDNS; or queries the mDNS status.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
:LAN:MDNS ON /*Enables mDNS.*/
:LAN:MDNS? /*The query returns 1.*/

## :LAN:APPLy

**Syntax**
:LAN:APPLy

**Description**
Applies the network configuration.

**Remarks**
After configuring the LAN parameters, run this command to apply the LAN settings.

## :LAN:HOST:NAME

**Syntax**
:LAN:HOST:NAME <name>
:LAN:HOST:NAME?

**Description**
Sets or queries the host name.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <name> | ASCII String | The label can contain English letters and numbers, as well as some symbols. | —— |

**Return Format**
The query returns the host name in ASCII strings.

## :LAN:DESCription

**Syntax**
:LAN:DESCription <name>
:LAN:DESCription?

**Description**
Sets or queries the description.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <name> | ASCII String | The label can contain English letters and numbers, as well as some symbols. | —— |

**Return Format**
The query returns the description in ASCII strings.

# :MMEMory Commands

## :MMEMory:DELete

**Syntax**
:MMEMory:DELete <file_name>

**Description**
Deletes a specified file.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name> | ASCII String | —— | -- |

**Remarks**
<file_name> should contain the path and the filename.
This operation fails if the file with the specified filename does not exist.

**Example**
The following command deletes the state1.sta file from the /VSA/state folder.
:MMEMory:DELete /VSA/state/state1.sta

## :MMEMory:LOAD:STATe

**Syntax**
:MMEMory:LOAD:STATe <file_name>

**Description**
Loads the specified state file suffixed with "*.sta".

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name> | ASCII String | —— | —— |

**Remarks**
This operation fails if the file with the specified filename does not exist.

**Example**
The following command loads the state file (state1.sta) to the instrument.
:MMEMory:LOAD:STATe state1.sta

## :MMEMory:LOAD:RESults

**Syntax**
:MMEMory:LOAD:RESults <label>,<path>

**Description**
Loads the measurement results.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <label> | Discrete | RAWDATA | —— |
| <path> | Real | —— | —— |

**Remarks**
This operation fails if the file with the specified filename does not exist.

**Example**
The following command loads the rawdata.csv measurement data file into the raw data.
:MMEMory:LOAD:RESults RAWDATA,rawdata1.csv

## :MMEMory:MOVE

**Syntax**
:MMEMory:MOVE <file_name1>,<file_name2>

**Description**
Renames the specified file <file_name1> as <file_name2>.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name1> | ASCII String | —— | —— |
| <file_name2> | ASCII String | —— | —— |

**Remarks**
<file_name1> and <file_name2> should contain the path and the filename.
This operation fails if the file with the specified filename does not exist.

**Example**
The following command renames the state file (state1.sta) in the folder (/VSA/state) as "state2.sta".
:MMEMory:MOVE /VSA/state/state1.sta, /VSA/state/state2.sta

## :MMEMory:STORe:IMG:PNG

**Syntax**
:MMEMory:STORe:IMG:PNG <path>

**Description**
Saves the screen image to the specified path.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <path> | ASCII String | The label can contain English letters and numbers, as well as some symbols. | —— |

**Remarks**
<path> should contain the path and the filename.

**Example**
The following command sets the saving path of the image to "/data/UserData/vsa/image.png".
:MMEMory:STORe:IMG:PNG /data/UserData/vsa/image.png

## :MMEMory:STORe:STATe

**Syntax**
:MMEMory:STORe:STATe <file_name>
:MMEMory:STORe:STATe?

**Description**
Saves the current instrument state with the specified filename suffixed with ".sta" to the default path (/"mode name"/state).

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name> | ASCII String | —— | —— |

**Remarks**
If the specified file already exists, overwrite it.

**Example**
The following command saves the current instrument state with the filename "state.sta" to the folder (/VSA/state).
:MMEMory:STORe:STATe state

The following command returns /VSA/state/state.sta.
:MMEMory:STORe:STATe?

## :MMEMory:STORe:RESults

**Syntax**
:MMEMory:STORe:RESults <file_name>

**Description**
Saves the current raw data with the specified filename suffixed with ".csv" by default (you do not have to add the suffix manually) to the default path (/"mode name"/measdata).

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name> | ASCII String | —— | —— |

**Remarks**
If the specified file already exists, overwrite it.

**Example**
The following command saves the current measurement results with the specified filename "data" to the folder (/VSA/measdata).
:MMEMory:STORe:RESults data

## :MMEMory:STORe:RETurn?

**Syntax**
:MMEMory:STORe:RETurn?

**Description**
Obtains the saved results.

## :MMEMory:STORe:SCReen

**Syntax**
:MMEMory:STORe:SCReen <file_name>

**Description**

Saves the current screen image with the specified filename suffixed with "*.jpg", "*.png", or "*.bmp" to the default path (/"mode name"/screen).

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <file_name> | ASCII String | —— | —— |

**Remarks**
If the specified file already exists, overwrite it.
If a suffix (.jpg/.png/.bmp) is added to the filename, you can save the current screen image with a different format based on its different suffix.
If no suffix is added to the filename, then by default, the current screen image is saved in the currently selected format.

**Example**
The following command saves the current screen image with the filename "screen.jpg" to the folder (/VSA/screen).
:MMEMory:STORe:SCReen screen.jpg

## :MMEMory:STORe:SCReen:DATA?

**Syntax**
:MMEMory:STORe:SCReen:DATA?

**Description**
Saves the screen data.

## :MMEMory:USER:STORe

**Syntax**
:MMEMory:USER:STORe <user>
:MMEMory:USER:STORe?

**Description**
Sets the preset settings.
Queries the preset settings.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <user> | Discrete | DEF\|USER1\|USER2\|USER3\|USER4\|USER5\|USER6 | DEF |

**Remarks**
N/A

**Example**
The following command selects USER1.
:MMEMory:USER:STORe USER1

The following query returns USER1.
:MMEMory:USER:STORe?

## :SAVE:MEASure

**Syntax**
:SAVE:MEASure <type>,<path>

**Description**
Saves the current measurement data with the specified filename suffixed with ".csv" by default (you do not have to add the suffix manually) to the default path (/"mode name"/measdata).

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <type> | Discrete | RESULT\|RAWDATA | —— |
| <path> | ASCII String | —— | —— |

**Remarks**
If the specified file already exists, overwrite it.
RESULT: measurement results.
RAWDATA: raw data.

**Example**
The following command saves the current measurement results with the file name measdata1 to the /VSA/measdata folder.
:SAVE:MEASure RESULT,measdata1

# [:SENSe] Commands

## [:SENSe]:FREQuency:CENTer

**Syntax**
[:SENSe]:FREQuency:CENTer <freq>
[:SENSe]:FREQuency:CENTer?

**Description**
Sets the center frequency in the Spectrum view.
Queries the center frequency in the Spectrum view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <freq> | Real | 2.402 GHz to 2.480 GHz | 2.402 GHz |

**Return Format**
The query returns the center frequency in real number. The unit is Hz.

**Example**
The following command sets the center frequency to 2.406 GHz.
:SENSe:FREQuency:CENTer 2406000000

The following query returns 2406000000.000000.
:SENSe:FREQuency:CENTer?

## [:SENSe]:FREQuency:CENTer:STEP:AUTO

**Syntax**
[:SENSe]:FREQuency:CENTer:STEP:AUTO <bool>
[:SENSe]:FREQuency:CENTer:STEP:AUTO?

**Description**
Enables or disables the auto setting mode of the CF step.
Queries the status of the auto setting mode of the CF step.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Return Format**
The query returns 0 or 1.

**Example**
The following command enables the auto setting mode of the CF step.
:SENSe:FREQuency:CENTer:STEP:AUTO ON or :SENSe:FREQuency:CENTer:STEP:AUTO 1

The following query returns 1.
:SENSe:FREQuency:CENTer:STEP:AUTO?

## [:SENSe]:FREQuency:CENTer:STEP[:INCRement]

**Syntax**
[:SENSe]:FREQuency:CENTer:STEP[:INCRement] <freq>
[:SENSe]:FREQuency:CENTer:STEP[:INCRement]?

**Description**
Sets the CF step.
Queries the CF step.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <freq> | Real | -$F_{max}$ to $F_{max}$ | 2 MHz |

**Remarks**
$F_{max}$ is the maximum frequency supported by the current instrument model.

**Return Format**
The query returns the CF step in real number. The unit is Hz.

**Example**
The following command sets the CF step to 100 kHz.
:SENSe:FREQuency:CENTer:STEP:INCRement 100000

The following query returns 100000.000000.
:SENSe:FREQuency:CENTer:STEP:INCRement?

## [:SENSe]:FREQuency:SPAN

**Syntax**
[:SENSe]:FREQuency:SPAN <freq>
[:SENSe]:FREQuency:SPAN?

**Description**
Sets the span in the Spectrum view.
Queries the span in the Spectrum view.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <freq> | Real | 10 Hz to (sample rate/1.27875) | 6.256109 MHz |

**Return Format**
The query returns the span in real number. Its unit is Hz.

**Example**
The following command sets the span to 2 MHz.
:SENSe:FREQuency:SPAN 2000000

The following query returns 2000000.000000.
:SENSe:FREQuency:SPAN?

## [:SENSe]:FREQuency:STARt

**Syntax**
[:SENSe]:FREQuency:STARt <freq>
[:SENSe]:FREQuency:STARt?

**Description**
Sets the start frequency in the Spectrum view.
Queries the start frequency in the Spectrum view.

**Parameter**

| Name | Type | Range[1] | Default |
|---|---|---|---|

| | | | |
|---|---|---|---|
| <freq> | Real | (2.4 GHz - Span Limit/2) to (2.4820 GHz - 10 Hz) | 2.40 GHz |

Note[1]: Span Limit = Sample Rate/1.27875

**Return Format**
The query returns the start frequency in real number. The unit is Hz.

**Example**
The following command sets the start frequency to 2.42 GHz.
:SENSe:FREQuency:STARt 2420000000

The following query returns 2420000000.000000.
:SENSe:FREQuency:STARt?

## [:SENSe]:FREQuency:STOP

**Syntax**
[:SENSe]:FREQuency:STOP <freq>
[:SENSe]:FREQuency:STOP?

**Description**
Sets the stop frequency in the Spectrum view.
Queries the stop frequency in the Spectrum view.

**Parameter**

| Name | Type | Range[1] | Default |
|---|---|---|---|
| <freq> | Real | (2.40 GHz + 10 Hz) to (2.4820 GHz + Span Limit/2) | 2.404 GHz |

Note[1]: Span Limit = Sample Rate/1.27875

**Return Format**
The query returns the stop frequency in real number. The unit is Hz.

**Example**
The following command sets the stop frequency to 2.48 GHz.
:SENSe:FREQuency:STOP 2480000000

The following query returns 2480000000.000000.
:SENSe:FREQuency:STOP?

## [:SENSe]:POWer[:RF]:ATTenuation

**Syntax**
[:SENSe]:POWer[:RF]:ATTenuation <integer>
[:SENSe]:POWer[:RF]:ATTenuation?

**Description**
Sets the attenuation of the RF front-end attenuator.
Queries the attenuation of the RF front-end attenuator.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <integer> | Integer | 0 dB to 50 dB | 10 dB |

**Return Format**
The query returns the attenuation in integer. The unit is dB.

**Example**

The following command sets the attenuation to 20 dB.
:SENSe:POWer:RF:ATTenuation 20

The following query returns 20.
:SENSe:POWer:RF:ATTenuation?

## [:SENSe]:TX:AVERage:COUNt

**Syntax**
[:SENSe]:TX:AVERage:COUNt <integer>
[:SENSe]:TX:AVERage:COUNt?

**Description**
Sets the trace average count of the current measurement.
Queries the trace average count of the current measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <integer> | Integer | 1 to 10000 | 100 |

**Return Format**
The query returns the number of average times in integer.

**Example**
The following command sets the average count to 100.
:SENSe:TX:AVERage:COUNt 100

The following query returns 100.
:SENSe:TX:AVERage:COUNt?

## [:SENSe]:TX:AVERage[:STATe]

**Syntax**
[:SENSe]:TX:AVERage[:STATe] <bool>
[:SENSe]:TX:AVERage[:STATe]?

**Description**
Sets the average state.
Queries the average state.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
The following command enables the average state.
:SENSe:TX:AVERage:STATe ON or :SENSe:TX:AVERage:STATe 1

The following query returns 1.
:SENSe:TX:AVERage:STATe?

## [:SENSe]:TX:AVERage:TYPE

**Syntax**
[:SENSe]:TX:AVERage:TYPE <enum>
[:SENSe]:TX:AVERage:TYPE?

**Description**
Selects the average type of the BLE analysis measurement.
Queries the average type of the BLE analysis measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | SCALar\|MIN\|MAX | SCALar |

**Remarks**
SCALar: Average (standard); MIN: Minimum; MAX: Maximum.

**Return Format**
The query returns SCAL, MIN, or MAX.

**Example**
The following command sets the average type of the BLE analysis measurement to MAX.
:SENSe:TX:AVERage:TYPE MAX

The following query returns MAX.
:SENSe:TX:AVERage:TYPE?

## [:SENSe]:TX:BANDwidth:RESolution

**Syntax**
[:SENSe]:TX:BANDwidth:RESolution?

**Description**
Queries the resolution bandwidth (RBW) of the BLE measurement.

**Return Format**
The query returns the RBW of BLE measurement in scientific notation. The unit is Hz.

**Example**
The following query returns 3.2012195122E+02.
:SENSe:TX:BANDwidth:RESolution?

## [:SENSe]:TX:BANDwidth:FFT:SHAPe

**Syntax**
[:SENSe]:TX:BANDwidth:FFT:SHAPe <enum>
[:SENSe]:TX:BANDwidth:FFT:SHAPe?

**Description**
Sets the FFT window type.
Queries the FFT window type.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | RECT\|HANN\|GAUS\|FLAT | RECT |

**Remarks**
RECT: Rectangular;HANN: Hanning; GAUS: Gaussian;FLAT: Flat Top.

**Return Format**
The query returns RECT, HANN, GAUS, or FLAT.

**Example**
The following command sets the FFT window type of the BLE analysis measurement to GAUS.
:SENSe:TX:BANDwidth:FFT:SHAPe GAUS

The following query returns GAUS.
:SENSe:TX:BANDwidth:FFT:SHAPe?

## [:SENSe]:TX:FREQuency:CHANnel:NUMBer

**Syntax**
[:SENSe]:TX:FREQuency:CHANnel:NUMBer <integer>
[:SENSe]:TX:FREQuency:CHANnel:NUMBer?

**Description**
Sets the channel number for BLE measurement.
Queries the channel number for BLE measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <integer> | Real | 0 to 39 | 0 |

**Return Format**
The query returns the channel number in scientific notation.

**Example**
The following command sets the channel number to 4.
:SENSe:TX:FREQuency:CHANnel:NUMBer 4

The following query returns 4.0E+00.
:SENSe:TX:FREQuency:CHANnel:NUMBer?

## [:SENSe]:TX:PACKet:TYPE

**Syntax**
[:SENSe]:TX:PACKet:TYPE <enum>
[:SENSe]:TX:PACKet:TYPE?

**Description**
Sets the packet type.
Queries the packet type.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | LE1M\|LE2M\|LE500k\|LE125k | LE1M |

**Return Format**
The query returns LE1M, LE2M, LE500k, or LE125k.

**Example**
The following command sets the packet type of the BLE analysis measurement to LE2M.
:SENSe:TX:PACKet:TYPE LE2M

The following query returns LE2M.
:SENSe:TX:PACKet:TYPE?

## [:SENSe]:RADio:STANdard

**Syntax**
[:SENSe]:RADio:STANdard <enum>
[:SENSe]:RADio:STANdard?

**Description**
Sets the standard type of BLE measurement.
Queries the standard type of BLE measurement.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | LENergy | LENergy |

**Remarks**
Only LENergy (low energy) is supported.

**Return Format**
The query returns LEN.

**Example**
The following command sets the standard type of BLE measurement to LEN.
:SENSe:RADio:STANdard LENergy

The following query returns LEN.
:SENSe:RADio:STANdard?

## [:SENSe]:SAMPle:RATE

**Syntax**
[:SENSe]:SAMPle:RATE <rate>
[:SENSe]:SAMPle:RATE?

**Description**
Sets the sample rate.
Queries the sample rate.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <rate> | Real | Refer to Remarks | 8 MHz |

**Remarks**
When the packet type is LE 1M, LE 500k, or LE 125k, the value of the parameter <rate> ranges from 4 MHz to 32 MHz.
When the packet type is set to LE 2M, the range of the parameter <rate> is from 8 MHz to 64 MHz.

**Return Format**
The query returns the sample rate in scientific notation. The unit is Hz.

**Example**
The following command sets the sample rate to 20 MHz.
:SENSe:SAMPle:RATE 20000000

The following query returns 2.0E+07.

:SENSe:SAMPle:RATE?

## [:SENSe]:ACQuisition:TIME

**Syntax**
[:SENSe]:ACQuisition:TIME <time>
[:SENSe]:ACQuisition:TIME?

**Description**
Sets the acquisition time.
Queries the acquisition time.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <time> | Real | Refer to Remarks | 3 ms |

**Remarks**
When the packet type is LE 1M, LE 500k, or LE 125k, the value of the parameter <time> ranges from 100 μs to 20 ms.
When the packet type is set to LE 2M, the range of the parameter <time> is from 100 μs to 10 ms.

**Return Format**
The query returns the acquisition time in scientific notation. The unit is s.

**Example**
The following command sets the acquisition time to 5 ms.
:SENSe:ACQuisition:TIME 0.005

The following query returns 5.00E-03.
:SENSe:ACQuisition:TIME?

## [:SENSe]:CORRection:TX:STATe

**Syntax**
[:SENSe]:CORRection:TX:STATe <bool>
[:SENSe]:CORRection:TX:STATe?

**Description**
Enables or disables the amplitude correction.
Queries the on/off status of the amplitude correction.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | ON\|1 |

**Return Format**
The query returns 0 or 1.

**Example**
The following command enables the amplitude correction.
:SENSe:CORRection:TX:STATe ON or :SENSe:CORRection:TX:STATe 1

The following query returns 1.
:SENSe:CORRection:TX:STATe?

## [:SENSe]:CORRection:TX[:RF]:GAIN

**Syntax**
[:SENSe]:CORRection:TX[:RF]:GAIN <rel_ampl>
[:SENSe]:CORRection:TX[:RF]:GAIN?

**Description**
Sets the external gain.
Queries the external gain.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <rel_ampl> | Real | -120 dB to 120 dB | 0 dB |

**Return Format**
The query returns the external gain value in scientific notation. The unit is dB.

**Example**
The following command sets the external gain value to 20 dB.
:SENSe:CORRection:TX:RF:GAIN 20

The following query returns 2.0E+01.
:SENSe:CORRection:TX:RF:GAIN?

## [:SENSe]:CORRection:IMPedance[:INPut][:MAGNitude]

**Syntax**
[:SENSe]:CORRection:IMPedance[:INPut][:MAGNitude] <enum>
[:SENSe]:CORRection:IMPedance[:INPut][:MAGNitude]?

**Description**
Sets the input impedance. Its unit is Ω.
Queries the input impedance.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | 50\|75 | 50 |

**Remarks**
If the output impedance of the system under test is 75 Ω, you should use a 75 Ω to 50 Ω adapter (option) supplied by **RIGOL** to connect the analyzer with the system under test, and then set the input impedance to 75 Ω.

**Return Format**
The query returns 50 or 75.

**Example**
The following command sets the input impedance to 75 Ω.
:SENSe:CORRection:IMPedance:INPut:MAGNitude 75

The following query returns 75.
:SENSe:CORRection:IMPedance:INPut:MAGNitude?

# :SYSTem Commands

## :SYSTem:BEEPer

**Syntax**
:SYSTem:BEEPer <bool>
:SYSTem:BEEPer?

**Description**
Enables or disables the beeper; queries the on/off status of the beeper.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
:SYSTem:BEEPer ON or :SYSTem:BEEPer 1 /*Enables the beeper.*/
:SYSTem:BEEPer? /*The query returns 1.*/

## :SYSTem:BRIGhtness

**Syntax**
:SYSTem:BRIGhtness <brightness>
:SYSTem:BRIGhtness?

**Description**
Sets or queries the screen brightness.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <brightness> | Integer | 0% to 100% | 80% |

**Return Format**
The query returns the screen brightness in integer.

**Example**
:SYSTem:BRIGhtness 50 /*Sets the screen brightness to 50%.*/
:SYSTem:BRIGhtness? /*The query returns 50.*/

## :SYSTem:DATE

**Syntax**
:SYSTem:DATE <year>,<month>,<day>
:SYSTem:DATE?

**Description**
Sets or queries the system date of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <year> | ASCII String | 2000 to 2099 | —— |
| <month> | ASCII String | 01 to 12 | —— |

| | | | |
|---|---|---|---|
| <day> | ASCII String | 01 to 31 | —— |

**Return Format**
The query returns the current system date in the format of "YYYY-MM-DD".

**Example**
:SYSTem:DATE 2017,11,16 /*Sets the system date of the instrument to Nov. 16th, 2017.*/
:SYSTem:DATE? /*The query returns 2017-11-16.*/

## :SYSTem:GPIB

**Syntax**
:SYSTem:GPIB <adr>
:SYSTem:GPIB?

**Description**
Sets or queries the GPIB address.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <adr> | Integer | 1 to 30 | 1 |

**Return Format**
The query returns an integer ranging from 1 to 30.

**Example**
:SYSTem:GPIB 2 /*Sets the GPIB address to 2.*/
:SYSTem:GPIB? /*The query returns 2.*/

## :SYSTem:LANGuage

**Syntax**
:SYSTem:LANGuage <enum>

**Description**
Sets or queries the system language of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <enum> | Discrete | SCHinese\|TCHinese\|KORean\|JAPanese\|ENGLish\|GERMan\|PORTuguese\|POLish\|FRENch\|RUSSian\|SPAN\|THAI\|INDonesian | ENGLish |

**Remarks**
SCHinese: Simplified Chinese.
TCHinese: Traditional Chinese.
KORean: Korean.
JAPanese: Japanese.
ENGLish: English.
GERMan: German.
PORTuguese: Portuguese.
POLish: Polish.
FRENch: French.
RUSSian: Russian.
SPAN: Spanish.
THAI: Thai.
INDonesian: Indonesian.

**Return Format**
The query returns SCH, TCH, KOR, JAP, ENGL, GERM, PORT, POL, FREN, RUSS, SPAN, THAI, or IND.

**Example**
:SYSTem:LANGuage ENGLish /*Sets the system language to ENGLish.*/
:SYSTem:LANGuage? /*The query returns ENGL.*/

## :SYSTem:LOCKed

**Syntax**
**:SYSTem:LOCKed<bool>**
**:SYSTem:LOCKed?**

**Description**
Locks or unlocks the keypad.
Queries whether the keypad is locked or not.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
:SYSTem:LOCKed 1 or :SYSTem:LOCKed ON /*Disables the touch screen operation.*/
:SYSTem:LOCKed? /*The query returns 1.*/

## :SYSTem:OPTion:INSTall

**Syntax**
:SYSTem:OPTion:INSTall <option info>@<license info>

**Description**
Installs and activates the specified option.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <option info> | ASCII String | RSA6000-BLE_TX | RSA6000-BLE_TX |
| <license info> | ASCII String | —— | —— |

**Remarks**
The parameter <option info> indicates the order number of the option. <license info> indicates the serial number of the option.

**Example**
The following command installs the option RSA6000-BLE_TX.
:SYSTem:OPTion:INSTall RSA6000-BLE_TX
@8AD12B8EBC5DF492D1D4289B7CBA5B6150BF6F5D752D645C36D74530B05F39B49C461B23A50D6C94A34E06782AC4380070B0D1A86BA84E02768391FFD70C2103

## :SYSTem:OPTion:STATus?

**Syntax**

:SYSTem:OPTion:STATus? <option name>

**Description**
Queries whether the specified option is activated.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <option name> | Discrete | BND\|PA\|P08\|P14\|P26\|B200\|RB200\|<br>EMI\|T08\|ADM\|AWK\|VSA\|UBW\|PNM\|PMA\|LORA\|ZIGBEE\|BLE_TX | —— |

**Return Format**
The query returns 0 (not activated) or 1 (activated).

**Example**
:SYSTem:OPTion:STATus? RSA6000-BLE_TX /*The query returns 1, indicating that the RSA6000-BLE_TX option is activated.*/

## :SYSTem:OPTion:UNINstall

**Syntax**
:SYSTem:OPTion:UNINstall

**Description**
Uninstalls all the official options.

## :SYSTem:PON

**Syntax**
:SYSTem:PON <power_on>
:SYSTem:PON?

**Description**
Sets or queries the setting type when the instrument is powered on.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <power_on> | Discrete | DEFault\|LATest | PRESet |

**Remarks**
DEFault: indicates preset settings, including factory mode and 6 user-defined settings.
LATest: last settings.

**Return Format**
The query returns DEF or LAT.

**Example**
:SYSTem:PON LAT /*Sets the instrument to recall last settings at next power-on.*/
:SYSTem:PON? /*The query returns LAT.*/

## :SYSTem:RESet

**Syntax**
:SYSTem:RESet

**Description**
Restarts the instrument.

## :SYSTem:PRESet

**Syntax**
:SYSTem:PRESet

**Description**
Initializes the instrument.

## :SYSTem:PSTatus

**Syntax**
:SYSTem:PSTatus <status>
:SYSTem:PSTatus?

**Description**
Sets or queries the front-panel power switch status.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <status> | Discrete | DEFault\|OPEN | DEFault |

**Remarks**
DEFault: switches off.
OPEN: switches on.

**Return Format**
The query returns DEF or OPEN.

**Example**
:SYSTem:PSTatus OPEN /*Sets the power switch to be on.*/
:SYSTem:PSTatus? /*The query returns OPEN.*/

## :SYSTem:PWRD

**Syntax**
:SYSTem:PWRD

**Description**
Powers off the instrument.

## :SYSTem:STIME

**Syntax**
:SYSTem:STIME <bool>
:SYSTem:STIME?

**Description**
Sets or queries whether to display the system time.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 0 or 1.

**Example**
:SYSTem:STIME 1 or :SYSTem:STIME ON /*Sets to display the system date.*/
:SYSTem:STIME?  /*The query returns 1.*/

## :SYSTem:TIME

**Syntax**
:SYSTem:TIME <hour>,<minute>,<second>
:SYSTem:TIME?

**Description**
Sets or queries the system time of the instrument.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <hour> | ASCII String | 00 to 23 | —— |
| <minute> | ASCII String | 00 to 59 | —— |
| <second> | ASCII String | 00 to 59 | —— |

**Return Format**
The query returns the current system time in the format of "HH:MM:SS".

**Example**
:SYSTem:TIME 15,10,30  /*Sets the system time of the instrument to 15:10:30.*/
:SYSTem:TIME?  /*The query returns 15:10:30.*/

## :SYSTem:VERSion?

**Syntax**
:SYSTem:VERSion?

**Description**
Queries the version number of the SCPI used by the system.

# :TRIGger Commands

## :TRIGger[:SEQuence]:SOURce

**Syntax**
:TRIGger[:SEQuence]:SOURce <type>
:TRIGger[:SEQuence]:SOURce?

**Description**
Sets the trigger source.
Queries the trigger source.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <type> | Discrete | IMMediate\|POWer | IMMediate |

**Remarks**
IMMediate: indicates the free-run trigger.
POWer: indicates the IF power trigger.

**Return Format**
The query returns IMM or POW.

**Example**
The following command sets the trigger source to IF Power.
:TRIGger:SEQuence:SOURce POWer

The following query returns POW.
:TRIGger:SEQuence:SOURce?

## :TRIGger[:SEQuence]:ATRigger

**Syntax**
:TRIGger[:SEQuence]:ATRigger <time>
:TRIGger[:SEQuence]:ATRigger?

**Description**
Sets the time that the analyzer will wait for the trigger to be initiated automatically.
Queries the time that the analyzer will wait for the trigger to be initiated automatically.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <time> | Real | 1 ms to 100 s | 100 ms |

**Remarks**
This command is only valid when the auto triggering function is enabled.

**Return Format**
The query returns the time value in scientific notation. The unit is s.

**Example**
The following command sets the time to 10 ms.
:TRIGger:SEQuence:ATRigger 0.01

The following query returns 1.0E-02.
:TRIGger:SEQuence:ATRigger?

## :TRIGger[:SEQuence]:ATRigger:STATe

**Syntax**
:TRIGger[:SEQuence]:ATRigger:STATe <bool>
:TRIGger[:SEQuence]:ATRigger:STATe?

**Description**
Enables or disables the auto trigger function.
Queries the setting status of auto trigger function.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 1 or 0.

**Example**
The following command enables the auto trigger function.
:TRIGger:SEQuence:ATRigger:STATe ON or :TRIGger:SEQuence:ATRigger:STATe 1

The following query returns 1.
:TRIGger:SEQuence:ATRigger:STATe?

## :TRIGger[:SEQuence]:IF:DELay

**Syntax**
:TRIGger[:SEQuence]:IF:DELay <time>
:TRIGger[:SEQuence]:IF:DELay?

**Description**
Sets the delay time for the IF power trigger.
Queries the delay time for the IF power trigger.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <time> | Real | 0 to 500 ms | 1 μs |

**Remarks**
This command is only valid when the IF power trigger function is enabled.

**Return Format**
The query returns the delay time for IF power trigger in scientific notation. The unit is s.

**Example**
The following command sets the delay time for the IF power trigger to 10 ms.
:TRIGger:SEQuence:IF:DELay 0.01

The following query returns 1.0E-02.
:TRIGger:SEQuence:IF:DELay?

## :TRIGger[:SEQuence]:IF:DELay:STATe

**Syntax**
:TRIGger[:SEQuence]:IF:DELay:STATe <bool>
:TRIGger[:SEQuence]:IF:DELay:STATe?

**Description**
Sets the delay state for the IF power trigger.
Queries the delay state for the IF power trigger.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 1 or 0.

**Example**
The following command enables the delay function for the IF power trigger.
:TRIGger:SEQuence:IF:DELay:STATe ON or :TRIGger:SEQuence:IF:DELay:STATe 1

The following query returns 1.
:TRIGger:SEQuence:IF:DELay:STATe?

## :TRIGger[:SEQuence]:IF:LEVel

**Syntax**
:TRIGger[:SEQuence]:IF:LEVel <level>
:TRIGger[:SEQuence]:IF:LEVel?

**Description**
Sets the trigger level of the IF power trigger.
Queries the trigger level of the IF power trigger.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <level> | Real | -200 dBm to 100 dBm | -20 dBm |

**Remarks**
This command is only valid when the IF power trigger function is enabled.

**Return Format**
The query returns IF trigger level in scientific notation. The unit is dBm.

**Example**
The following command sets the trigger level of IF power trigger to 50 dBm.
:TRIGger:SEQuence:IF:LEVel 50

The following query returns 5.0E+01.
:TRIGger:SEQuence:IF:LEVel?

## :TRIGger[:SEQuence]:HOLDoff

**Syntax**
:TRIGger[:SEQuence]:HOLDoff <time>
:TRIGger[:SEQuence]:HOLDoff?

**Description**
Sets the trigger holdoff time.
Queries the trigger holdoff time.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <time> | Real | 100 us to 500 ms | 100 ms |

**Remarks**
This command is only valid when the trigger holdoff function is enabled.

**Return Format**
The query returns the trigger holdoff time in scientific notation. The unit is s.

**Example**
The following command sets the sync holdoff time to 100 ms.
:TRIGger:SEQuence:HOLDoff 0.1

The following query returns 1.0E-01.
:TRIGger:SEQuence:HOLDoff?

## :TRIGger[:SEQuence]:HOLDoff:STATe

**Syntax**
:TRIGger[:SEQuence]:HOLDoff:STATe <bool>
:TRIGger[:SEQuence]:HOLDoff:STATe?

**Description**
Enables or disables the trigger holdoff function.
Queries the status of the trigger holdoff function.

**Parameter**

| Name | Type | Range | Default |
|---|---|---|---|
| <bool> | Bool | OFF\|ON\|0\|1 | OFF\|0 |

**Return Format**
The query returns 1 or 0.

**Example**
The following command enables the trigger holdoff function.
:TRIGger:SEQuence:HOLDoff:STATe ON or :TRIGger:SEQuence:HOLDoff:STATe 1

The following query returns 1.
:TRIGger:SEQuence:HOLDoff:STATe?

# Chapter 3 Programming Examples

This chapter lists some programming examples to illustrate how to use commands to realize the common functions of the spectrum analyzer in the development environments such as Visual C++ 6.0, Visual Basic 6.0, and LabVIEW 2010. Also, the chapter lists some examples to illustrate how to control the spectrum analyzer to realize the common functions in Linux operating system. These examples are programmed based on NI-VISA library.

NI-VISA (National Instrument-Virtual Instrument Software Architecture), developed by NI (National Instrument), provides an advanced programming interface to communicate with various instruments through their bus lines. NI-VISA enables you to realize the communication between the analyzer and PC through instrument buses (such as USB). VISA defines a set of software commands, with which users can control the instrument without knowing the working principle of the interface buses. For details, refer to the help information of NI-VISA.

# Programming Instructions

This section introduces the problems that might occur during the programming process as well as their solutions. If these problems occur, please resolve them according to the corresponding instructions.

1. When you build a working environment via the network, it is recommended that you build a pure local area network.

2. If the local area network environment is complicated (e.g. many devices and broadcast messages exist), it is recommended that you add some fault tolerance during the programming process. For the details, refer to the instrument write/read operations with exception handling functions "InstrWriteEx()" and "InstrReadEx()" in "*Visual C++ 6.0 Programming Example*".

3. The Socket programming port No. of this device is 5555.

# Programming Preparations

The programming preparations introduced here are only applicable to programming by using Visual C++ 6.0, Visual Basic 6.0, and LabVIEW 2010 development tools in Windows operating system. For the preparations of programming in Linux operating system, refer to "*Linux Programming Example*" in "*Programming Preparations*".

First, check whether your PC has installed NI's VISA library. If not, download it from *http://www.ni.com/visa/*. In this manual, the default installation path is C:\Program Files\IVI Foundation\VISA.
Connect spectrum analyzer to the PC via the USB interface of the analyzer. Use the USB cable to connect the USB DEVICE interface on the rear panel of the analyzer to the USB interface of the PC.

After the analyzer is connected to the PC properly, start the analyzer. In this case, "Found New Hardware Wizard" dialog box appears on the PC. Please install "USB Test and Measurement Device (IVI)" according to the wizard.

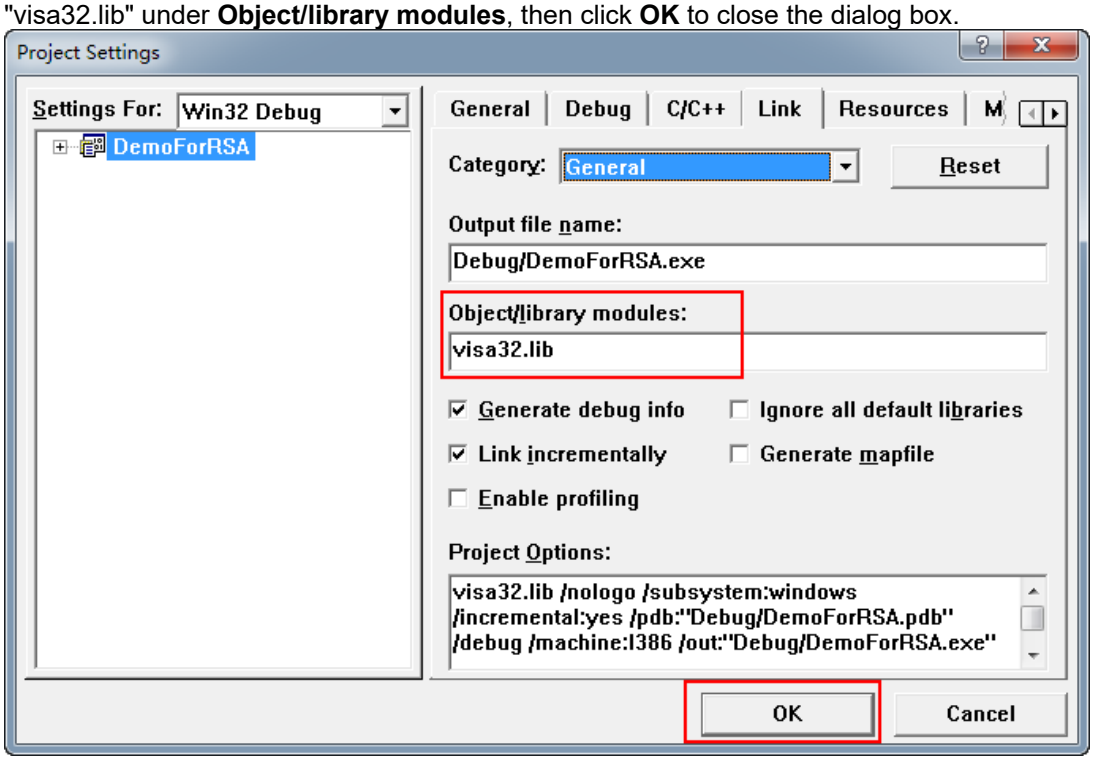
By now, the programming preparations are complete. The following parts will make a detailed introduction about the programming instances in the Visual C++ 6.0, Visual Basic 6.0, and LabVIEW 2010 development environment.

# Visual C++ 6.0 Programming Example

Enter the Visual C++6.0 programming environment, and perform the following procedures.

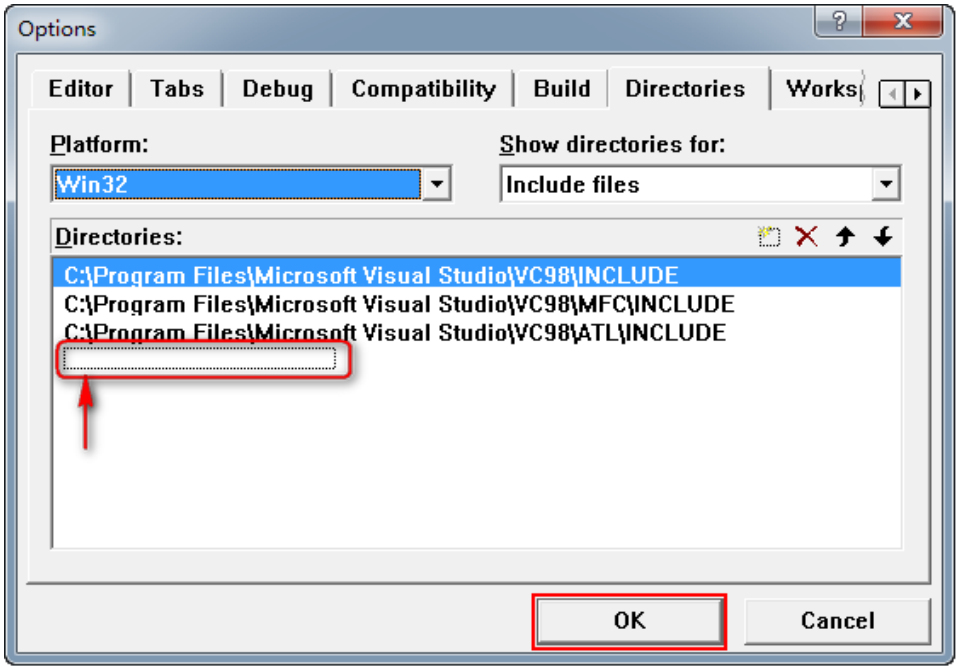
1. Create a MFC project based on a dialog box and name it "DemoForDSA" in this example.

2. Click **Project** → **Settings** to open the Project Setting dialog box. In the dialog box, click the **Link** tab, add "visa32.lib" under **Object/library modules**, then click **OK** to close the dialog box.



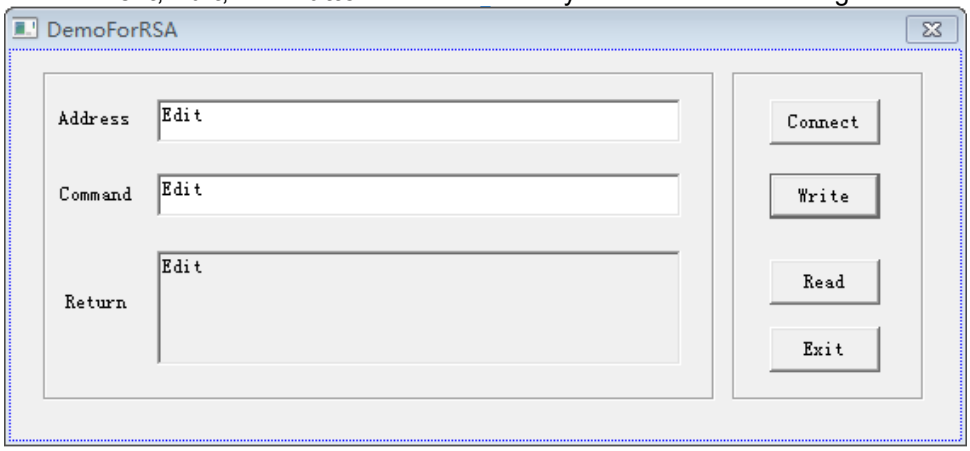

3. Click **Tools** → **Options** to open the Options dialog box. Then click the **Directories** tab.
   Select Include files from the drop-down list under **Show directories for**. Double click the empty space under **Directories** to enter the specified path of Include files: C:\Program Files\IVI Foundation\VISA\WinNT\include. Click **OK** to close the dialog box.
   Select Library files from the drop-down list under Show directories for. Double click the empty space under Directories to enter the specified path of Library files: C:\Program Files\IVI Foundation\VISA\WinNT\lib\msc. Click OK to close the dialog box.

   Note: The two paths added here are related to the installation path of NI-VISA on your PC. By default, NI-VISA is installed under C:\Program Files\IVI Foundation\VISA.




**By now, VISA library has been added.**

4. Add the **Text**, **Edit**, and **Button** controls. The layout interface for adding controls is as follows:




5. Add the control variables.
   Click **View → ClassWizard**, and then click on the "Member Variables" tab to add the following three variables:
   Instrument address: CString m_strInstrAddr
   Command: CString m_strCommand
   Returned value: CString m_strResult

6. Encapsulate the read and write operations of VISA.
   1) Encapsulate the write operation of VISA for easier operation.

```
bool CDemoForRSADlg::InstrWrite(CString strAddr, CString strContent) //Write operation
{
 ViSession defaultRM,instr;
 ViStatus status;
 ViUInt32 retCount;
 char * SendBuf = NULL;
 char * SendAddr = NULL;
 bool bWriteOK = false;
 CString str;
```

```
    // Change the address's data style from CString to char*
    SendAddr = strAddr.GetBuffer(strAddr.GetLength());
    strcpy(SendAddr,strAddr);
    strAddr.ReleaseBuffer();

    // Change the command's data style from CString to char*
    SendBuf = strContent.GetBuffer(strContent.GetLength());
    strcpy(SendBuf,strContent);
    strContent.ReleaseBuffer();

    //Open a VISA resource
    status = viOpenDefaultRM(&defaultRM);
    if (status < VI_SUCCESS)
    {
        AfxMessageBox("No VISA resource was opened!");
        return false;
     }

    status = viOpen(defaultRM, SendAddr, VI_NULL, VI_NULL, &instr);

    //Write command to the instrument
    status = viWrite(instr, (unsigned char *)SendBuf, strlen(SendBuf), &retCount);

    //Close the system
    status = viClose(instr);
    status = viClose(defaultRM);

    return bWriteOK;
  }
```

2) Encapsulate the read operation of VISA for easier operation.

```
  bool CDemoForRSADlg::InstrRead(CString strAddr, CString *pstrResult) //Read operation
  {
    ViSession defaultRM,instr;
    ViStatus status;
    ViUInt32 retCount;
    char * SendAddr = NULL;
    unsigned char RecBuf[MAX_REC_SIZE];
    bool bReadOK = false;
    CString str;

    // Change the address's data style from CString to char*
    SendAddr = strAddr.GetBuffer(strAddr.GetLength());
    strcpy(SendAddr,strAddr);
    strAddr.ReleaseBuffer();

    memset(RecBuf,0,MAX_REC_SIZE);

    //Open a VISA resource
    status = viOpenDefaultRM(&defaultRM);
    if (status < VI_SUCCESS)
    {
        // Error Initializing VISA...exiting
        AfxMessageBox("No VISA resource was opened!");
        return false;
     }

    //Open the instrument
    status = viOpen(defaultRM, SendAddr, VI_NULL, VI_NULL, &instr);
```

```
    //Read from the instrument
    status = viRead(instr, RecBuf, MAX_REC_SIZE, &retCount);

    //close the system
    status = viClose(instr);
    status = viClose(defaultRM);

    (*pstrResult).Format("%s",RecBuf);

    return bReadOK;
}
```

3) Encapsulate the read operation with exception handling function of VISA.

```
ViStatus CDemoForRSADlg::OpenVisaDevice(CString strAddr)      //Open a VISA device
{
    ViStatus status;
    char * SendAddr = NULL;

    // Change the address's data style from CString to char*
    SendAddr = strAddr.GetBuffer(strAddr.GetLength());
    strcpy(SendAddr,strAddr);
    strAddr.ReleaseBuffer();

    //Open a VISA resource
    status = viOpenDefaultRM(&m_SessRM);

    if (status == 0)
    {
        //Open the device
        status = viOpen(m_SessRM, SendAddr, VI_NULL, VI_NULL, &m_SessInstr);

        //If you fails to open the connection, close the resource
        if (status != 0)
        {
            viClose(m_SessRM);
        }
    }

    return status;
}


ViStatus CDemoForRSADlg::CloseVisaDevice()        //Close a VISA device
{
    ViStatus status;

    //Close the device
    status = viClose(m_SessInstr);

    if (status == 0)
    {
        //close the resource
        status = viClose(m_SessRM);
    }

    return status;
}


bool CDemoForRSADlg::InstrWriteEx(CString strAddr, CString strContent) //Write operation with
```

```
exception handling
{
ViStatus status;
ViUInt32 retCount;
char * SendBuf = NULL;
bool bWriteOK = true;

// Change the address's data style from CString to char*
SendBuf = strContent.GetBuffer(strContent.GetLength());
strcpy(SendBuf,strContent);
strContent.ReleaseBuffer();

do
{
      //Write command to the instrument
      status = viWrite(m_SessInstr, (unsigned char *)SendBuf, strlen(SendBuf), &retCount);

      //If an error occurs, perform error handing
      if (status < 0)
      {
            //If the time exceed the limit value, resend the command after a delay of 1s
            if (VI_ERROR_TMO == status)
            {
                  Sleep(1000);
                   status = viWrite(m_SessInstr, (unsigned char *)SendBuf, strlen(SendBuf), &retCount);
            }
            else
            {
                  //If another error occurs, reopen the connection after the connection is closed and
                  resend the command
                  status = CloseVisaDevice();
                  Sleep(1000);
                  status = OpenVisaDevice(m_strInstrAddr);
                  if (status == 0)
                  {
                        status = viWrite(m_SessInstr, (unsigned char *)SendBuf, strlen(SendBuf),
                        &retCount);
                   }
             }
       }
} while (status < 0);

return bWriteOK;
}


bool CDemoForRSADlg::InstrReadEx(CString strAddr, CString *pstrResult) //Read operation with
exception handling
{
ViStatus status;
ViUInt32 retCount;
char * SendAddr = NULL;
unsigned char RecBuf[MAX_REC_SIZE];
bool bReadOK = true;

// Change the address's data style from CString to char*
SendAddr = strAddr.GetBuffer(strAddr.GetLength());
strcpy(SendAddr,strAddr);
strAddr.ReleaseBuffer();
```

```
memset(RecBuf,0,MAX_REC_SIZE);

do
{
    //Read from the instrument
    status = viRead(m_SessInstr, RecBuf, MAX_REC_SIZE, &retCount);
    if (status < 0)
    {
        //If the time exceeds the limit value, read from the instrument after a delay of 1s
        if (VI_ERROR_TMO == status)
        {
            Sleep(1000);
            status = viRead(m_SessInstr, RecBuf, MAX_REC_SIZE, &retCount);
        }
        else
        {
            //If another error occurs, reopen the connection after the connection is closed and
            reread from instrument
            status = CloseVisaDevice();
            Sleep(1000);
            status = OpenVisaDevice(m_strInstrAddr);
            if (status == 0)
            {
                status = viRead(m_SessInstr, RecBuf, MAX_REC_SIZE, &retCount);
            }
        }
    }

} while (status < 0);


(*pstrResult).Format("%s",RecBuf);

return bReadOK;
}
```

7. Add the control message response codes.

1) Connect to the instrument

```
void CDemoForRSADlg::OnBtConnectInstr()              // Connect to the instrument
{
//TODO: Add your control notification handler code here
ViStatus status;
ViSession defaultRM;
ViString expr = "?*";
ViPFindList findList = new unsigned long;
ViPUInt32 retcnt = new unsigned long;
ViChar instrDesc[1000];
CString strSrc = "";
CString strInstr = "";
unsigned long i = 0;
bool bFindRSA = false;

status = viOpenDefaultRM(&defaultRM);
if (status < VI_SUCCESS)
{
// Error Initializing VISA...exiting
MessageBox("No VISA instrument was opened ! ");
return ;
}
```

```
        memset(instrDesc,0,1000);

        // Find resource
        status = viFindRsrc(defaultRM,expr,findList, retcnt, instrDesc);

        for (i = 0;i < (*retcnt);i++)
        {
            // Get instrument name
            strSrc.Format("%s",instrDesc);
            InstrWrite(strSrc,"*IDN?");
            ::Sleep(200);
            InstrRead(strSrc,&strInstr);

           // If the instrument(resource) belongs to the RSA series then jump out //from the loop
              strInstr.MakeUpper();
              if (strInstr.Find("RSA") >= 0)
              {
                  bFindRSA = true;
                  m_strInstrAddr = strSrc;
                  break;
              }

            //Find next instrument
            status = viFindNext(*findList,instrDesc);
        }

        if (bFindRSA == false)
        {
           MessageBox("Didn't find any RSA!");
        }
        UpdateData(false);
    }
```

2) Write Operation

```
    void CDemoForRSADlg::OnBtWrite()                    //Write operation
    {
     //TODO: Add your control notification handler code here
     UpdateData(true);
     if (m_strInstrAddr.IsEmpty())
     {
         MessageBox("Please connect to the instrument first!");
      }
     InstrWrite(m_strInstrAddr,m_strCommand);
     m_strResult.Empty();
     UpdateData(false);
    }
```

3) Read Operation

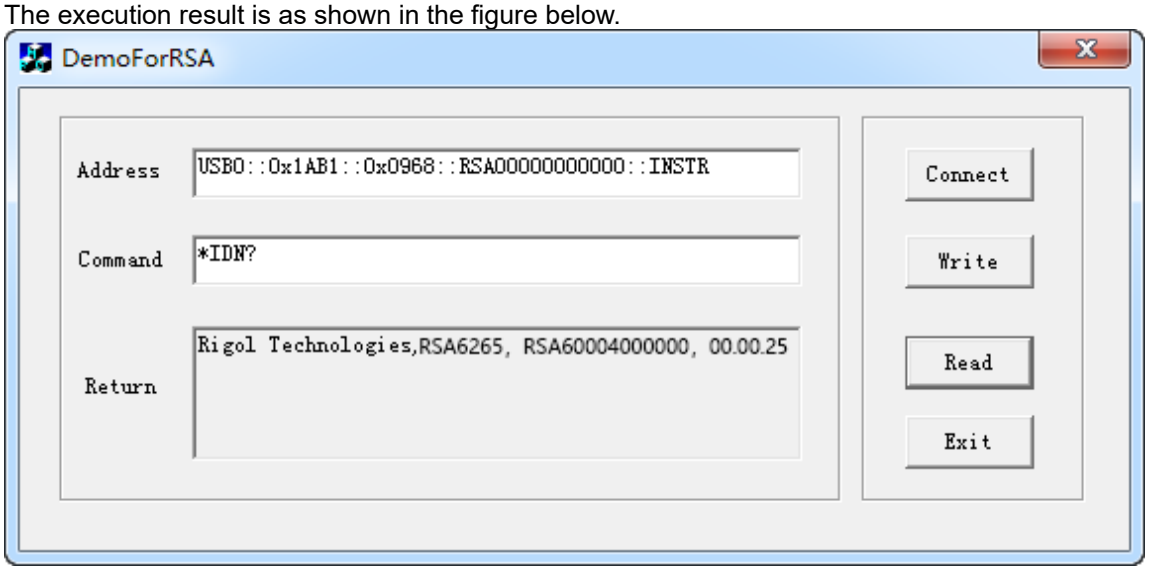
```
    void CDemoForRSADlg::OnBtRead()                     //Read operation
    {
     //TODO: Add your control notification handler code here
     UpdateData(true);
     InstrRead(m_strInstrAddr,&m_strResult);
     UpdateData(false);
    }
```

8. Run the results.

1) Click **Connect** to search for the spectrum analyzer;
2) Input "*IDN?" in the "Command" edit box;
3) Click **Write** to write the command into the spectrum analyzer;

4) Click **Read** to read the return value.

The execution result is as shown in the figure below.

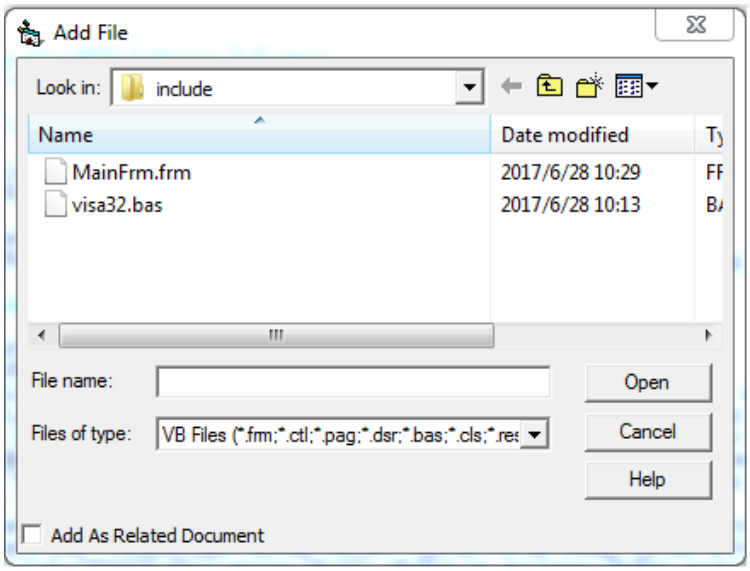


# Visual Basic 6.0 Programming Example

Enter the Visual Basic 6.0 programming environment, and perform the following procedures.

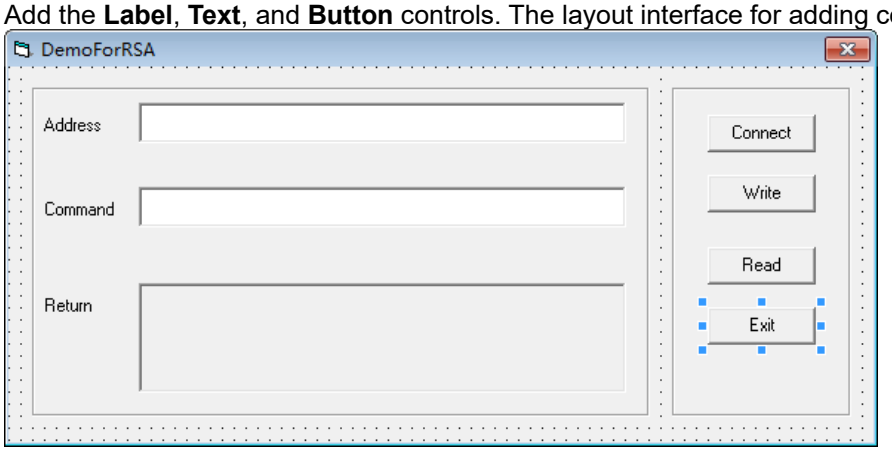
1. Build a standard application program project (Standard EXE), and name it "DemoForRSA".

2. Open **Project** → **Add File…**. Search for the visa32.bas file from the include folder in the installation path of NI-VISA, and then add the file to the project. The visa32.bas module contains all VISA functions and constant statements.




Then, add the Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long) statement into the visa32.bas module; or you can also create a new module to declare the Sleep function.

3. Add the **Label**, **Text**, and **Button** controls. The layout interface for adding controls is as follows:




4. Encapsulate the read and write operations of VISA.
   1) Encapsulate the write operation of VISA for easier operation.

```
'---------------------------------------------------------
'Function Name:  InstrWrite
'Function:  Send command to the instrument
'Input:  rsrcName,instrument(resource) name
         strCmd,Command
'---------------------------------------------------------
Public Sub InstrWrite(rsrcName As String, strCmd As String)
    Dim status As Long
    Dim dfltRM As Long
    Dim sesn As Long
    Dim rSize As Long
```

```
    'Initialize the system
    status = viOpenDefaultRM(dfltRM)
    'Failed to initialize the system
    If (status < VI_SUCCESS) Then
        MsgBox " No VISA resource was opened！ "
        Exit Sub
    End If
    'Open the VISA instrument
    status = viOpen(dfltRM, rsrcName, VI_NULL, VI_NULL, sesn)
    'Failed to open the instrument
    If (status < VI_SUCCESS) Then
        MsgBox "Failed to open the instrument！ "
        Exit Sub
    End If

    'Write command to the instrument
    status = viWrite(sesn, strCmd, Len(strCmd), rSize)
    'Failed to write to the instrument
    If (status < VI_SUCCESS) Then
        MsgBox " Faild to write to the instrument！ "
        Exit Sub
    End If

    'Close the system
    status = viClose(sesn)
    status = viClose(dfltRM)

End Sub
```

2) Encapsulate the read operation of VISA for easier operation.

```
'-------------------------------------------------------------
'Function Name： InstrRead
'Function： Read the return value from the instrument
'Input： rsrcName,Resource name
'Return： The string gotten from the instrument
'-------------------------------------------------------------
Public Function InstrRead(rsrcName As String) As String
    Dim status As Long
    Dim dfltRM As Long
    Dim sesn As Long
    Dim strTemp0 As String * 256
    Dim strTemp1 As String
    Dim rSize As Long

    'Begin by initializing the system
    status = viOpenDefaultRM(dfltRM)
    'Initialization failed
    If (status < VI_SUCCESS) Then
        MsgBox " Failed to open the instrument! "
        Exit Function
    End If
    'Open the instrument
    status = viOpen(dfltRM, rsrcName, VI_NULL, VI_NULL, sesn)
    'Failed to open the instrument
    If (status < VI_SUCCESS) Then
        MsgBox " Failed to open the instrument! "
        Exit Function
    End If
```

```
    'Read from the instrument
    stasus = viRead(sesn, strTemp0, 256, rSize)
    'Reading failed
    If (status < VI_SUCCESS) Then
        MsgBox " Failed to read from the instrument! "
        Exit Function
    End If

    'Close the system
    status = viClose(sesn)
    status = viClose(dfltRM)

    'Remove the space at the end of the string
    strTemp1 = Left(strTemp0, rSize)
    InstrRead = strTemp1
End Function
```

5. Add the control event codes.

1) Connect to the instrument

```
'Connect to the instrument
Private Sub CmdConnect_Click()
    Const MAX_CNT = 200
    Dim status As Long
    Dim dfltRM As Long
    Dim sesn As Long
    Dim fList As Long
    Dim buffer As String * MAX_CNT, Desc As String * 256
    Dim nList As Long, retCount As Long
    Dim rsrcName(19) As String * VI_FIND_BUFLEN, instrDesc As String * VI_FIND_BUFLEN
    Dim i, j As Long
    Dim strRet As String
    Dim bFindRSA As Boolean

    'Initialize the system
    status = viOpenDefaultRM(dfltRM)
    'Initialization failed
    If (status < VI_SUCCESS) Then
        MsgBox " No VISA resource was opened ！ "
        Exit Sub
    End If

    'Find instrument resource
    Call viFindRsrc(dfltRM, "USB?*INSTR", fList, nList, rsrcName(0))
    'Get the list of the instruments (resources)
    strRet = ""
    bFindRSA = False
    For i = 0 To nList - 1
    'Get the instrument name
    InstrWrite rsrcName(i), "*IDN?"
    Sleep 200
    strRet = InstrRead(rsrcName(i))
    'Continuing searching for the resource until an RSA instrument is found
    strRet = UCase(strRet)
    j = InStr(strRet, "RSA")
    If (j >= 0) Then
       bFindRSA = True
       Exit For
    End If
```

```
    Call viFindNext(fList + i - 1, rsrcName(i))
    Next i
    'Display
    If (bFindRSA = True) Then
       TxtInsAddr.Text = rsrcName(i)
    Else
       TxtInsAddr.Text = ""
    End If
End Sub
```

2) Write Operation

```
'Write the command to the instrument
Private Sub CmdWrite_Click()
    If (TxtInsAddr.Text = "") Then
        MsgBox ("Please write the instrument address!  ")
    End If

    InstrWrite TxtInsAddr.Text, TxtCommand.Text
End Sub
```

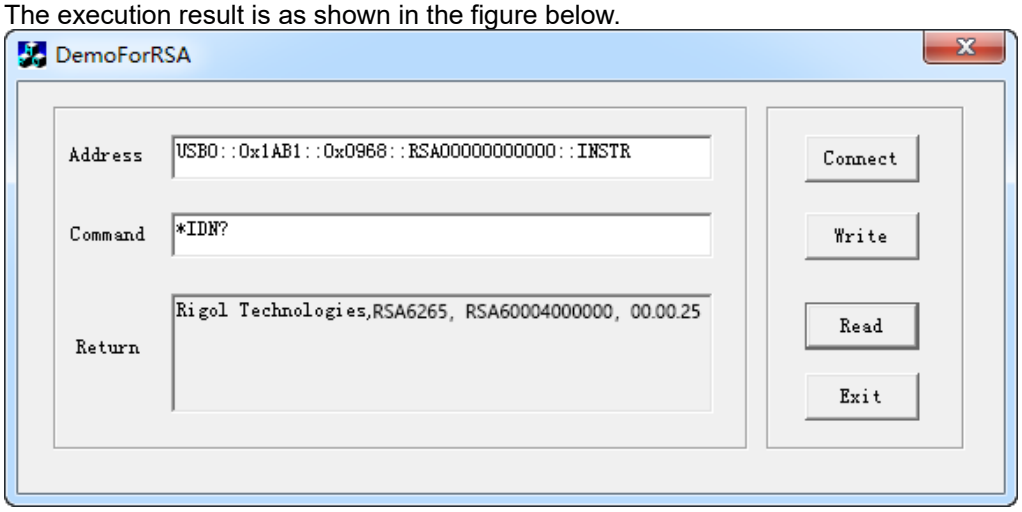
3) Read Operation

```
'Read the return value from the instrument
Private Sub CmdRead_Click()
    Dim strTemp As String
    strTemp = InstrRead(TxtInsAddr.Text)
    TxtReturn.Text = strTemp
End Sub
```

6. Run the results.

1) Click **Connect** to search for the spectrum analyzer;
2) Input "*IDN?" in the "Command" edit box;
3) Click **Write** to write the command into the spectrum analyzer;
4) Click **Read** to read the return value.

The execution result is as shown in the figure below.

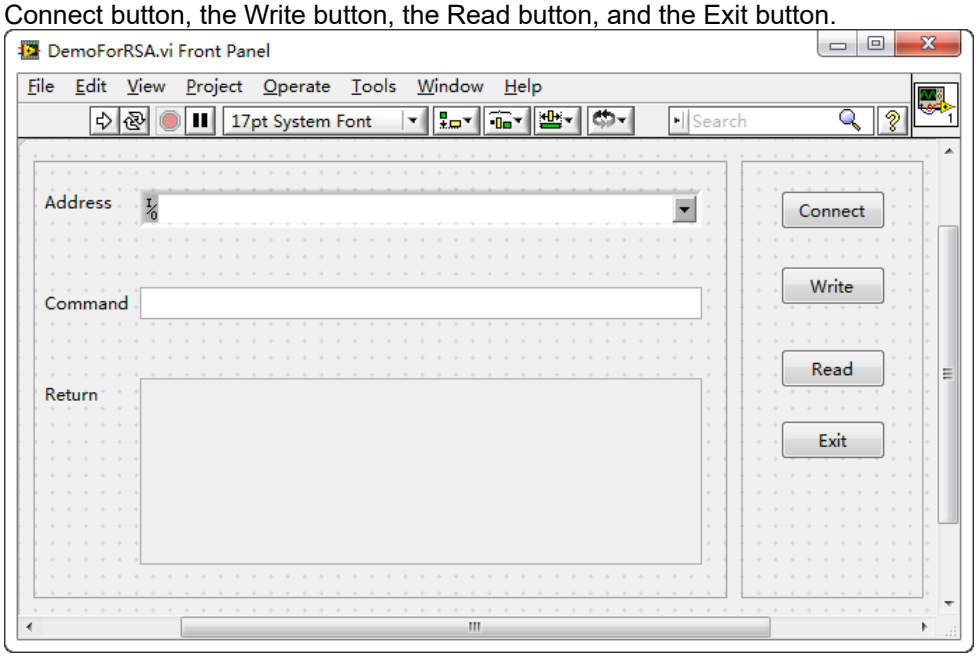


# LabVIEW 2010 Programming Example

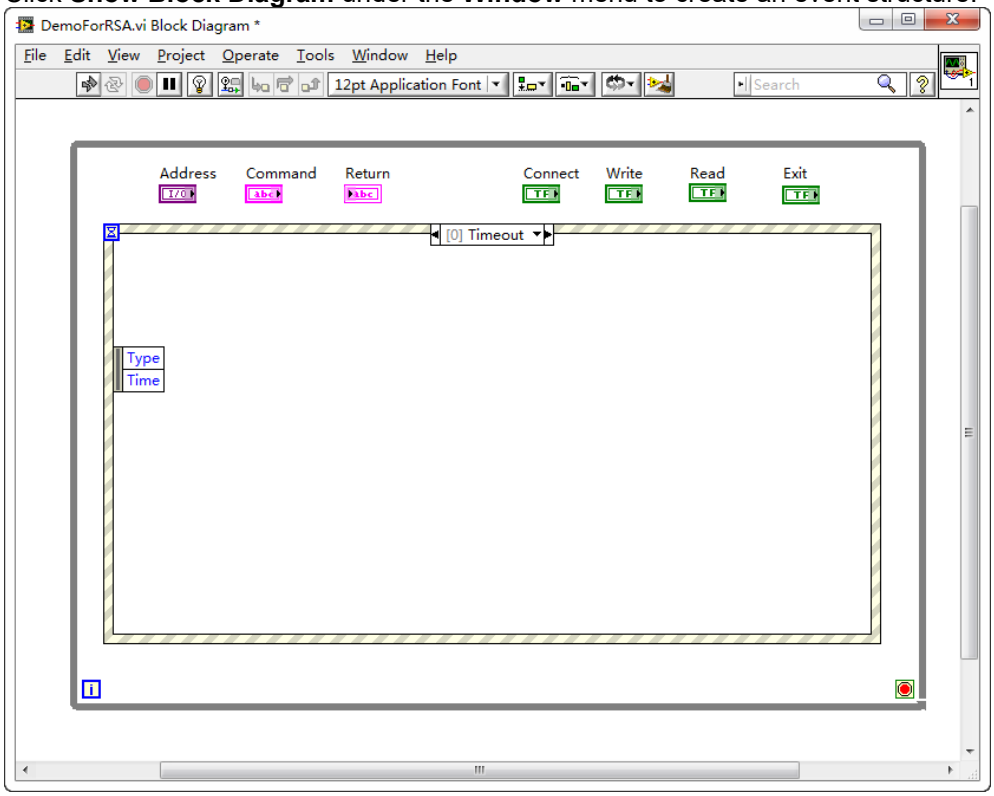
Enter the Labview 2010 programming environment, and perform the following procedures.

1. Create a VI file, and name it "DemoForRSA".

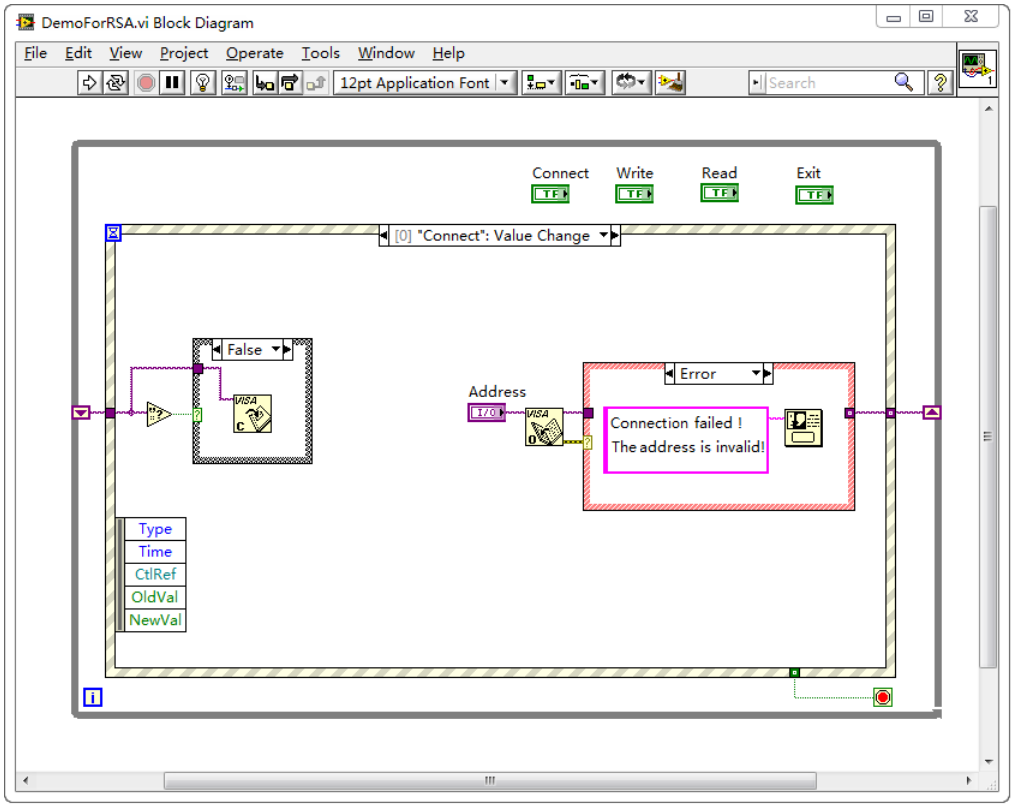
2. Add controls to the front panel interface, including the Address field, Command field, and Return field, the Connect button, the Write button, the Read button, and the Exit button.



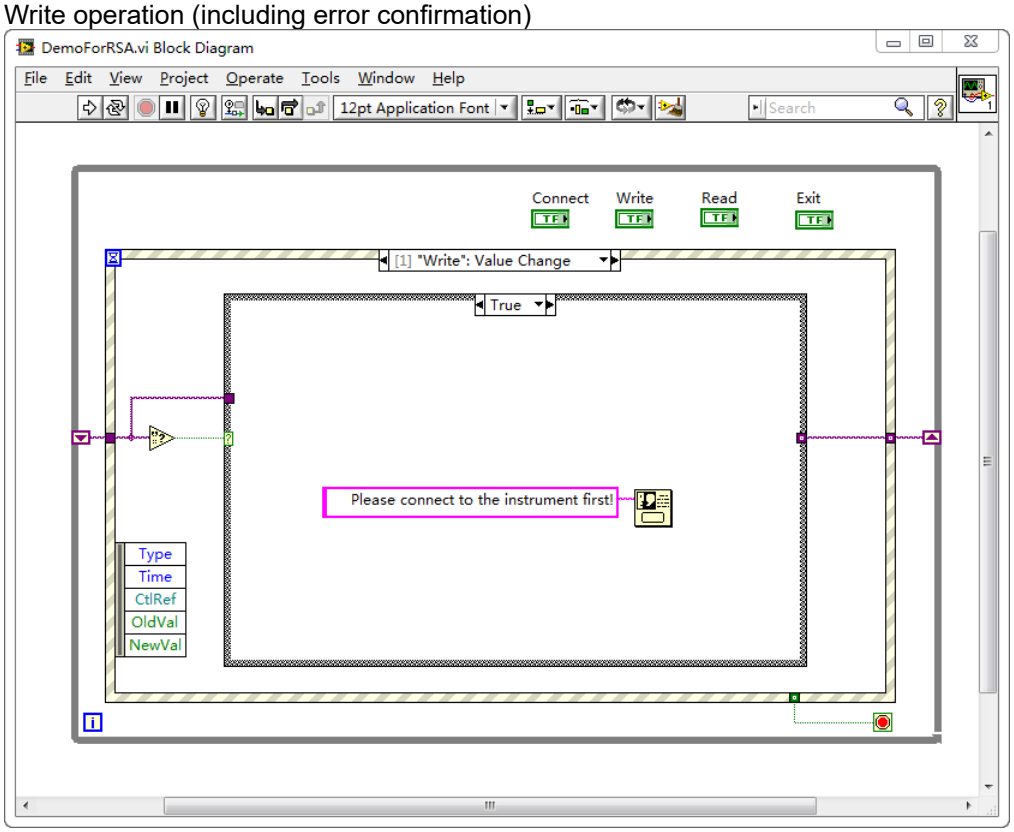

3. Click **Show Block Diagram** under the **Window** menu to create an event structure.

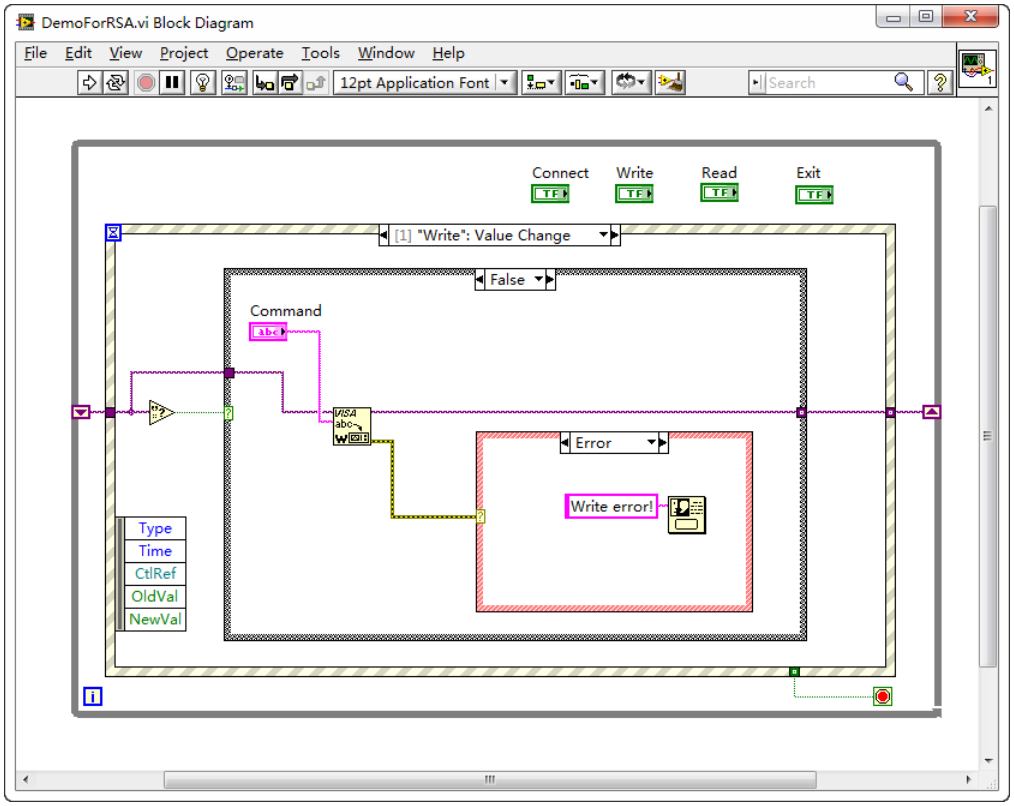



4. Add the events (including connecting to the instrument, write operation, read operation, and exit)
   1) Connect to the instrument

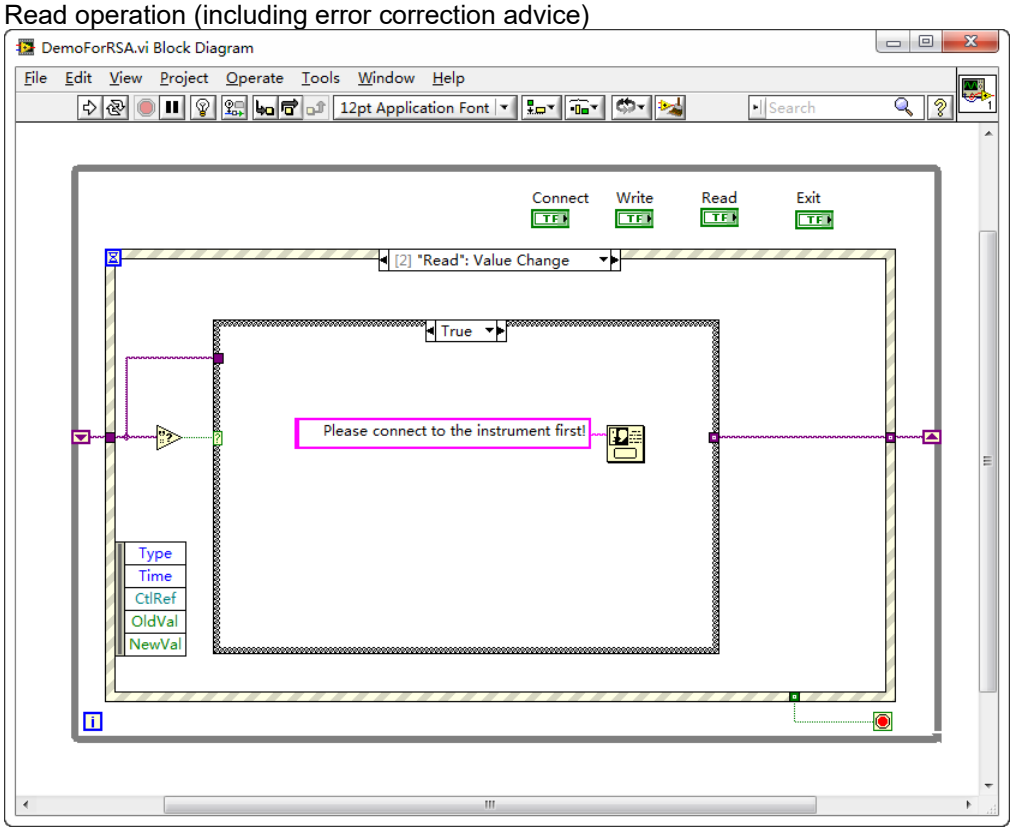


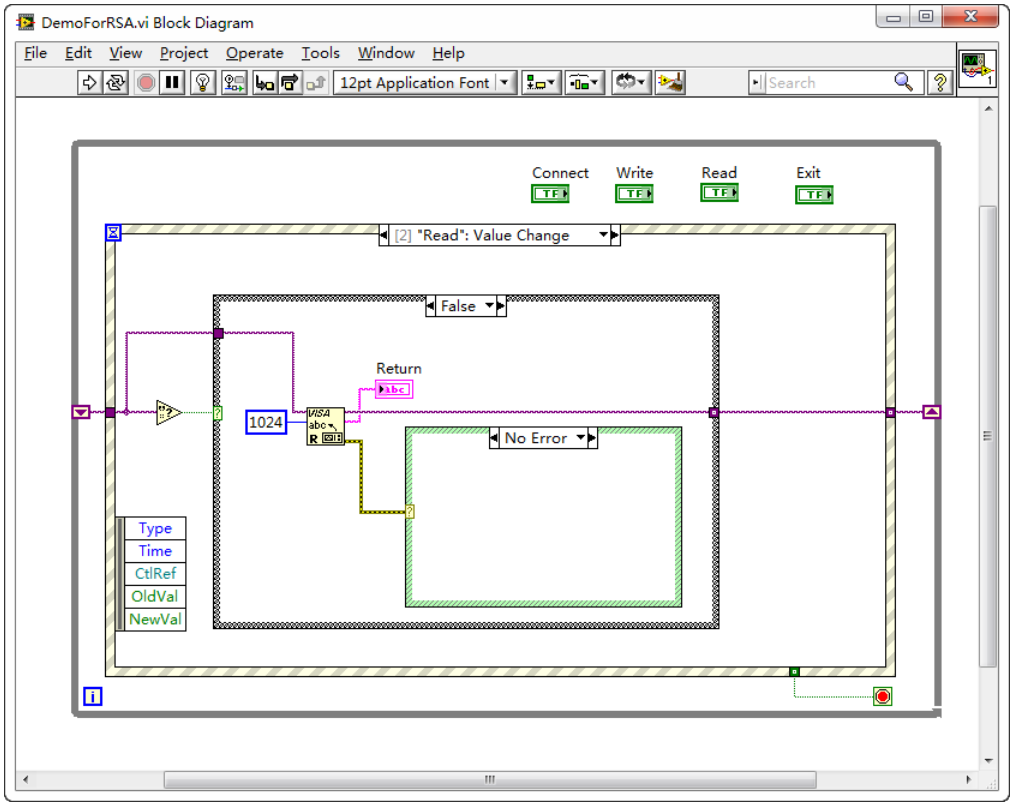

2) Write operation (including error confirmation)



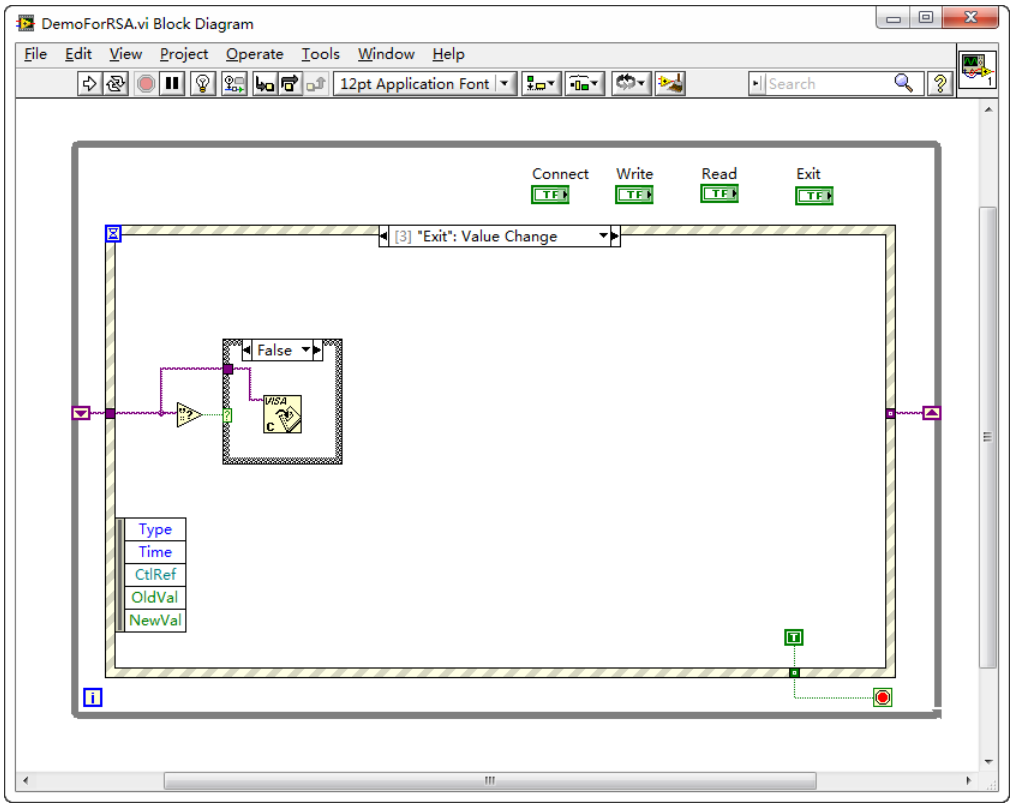



3) Read operation (including error correction advice)

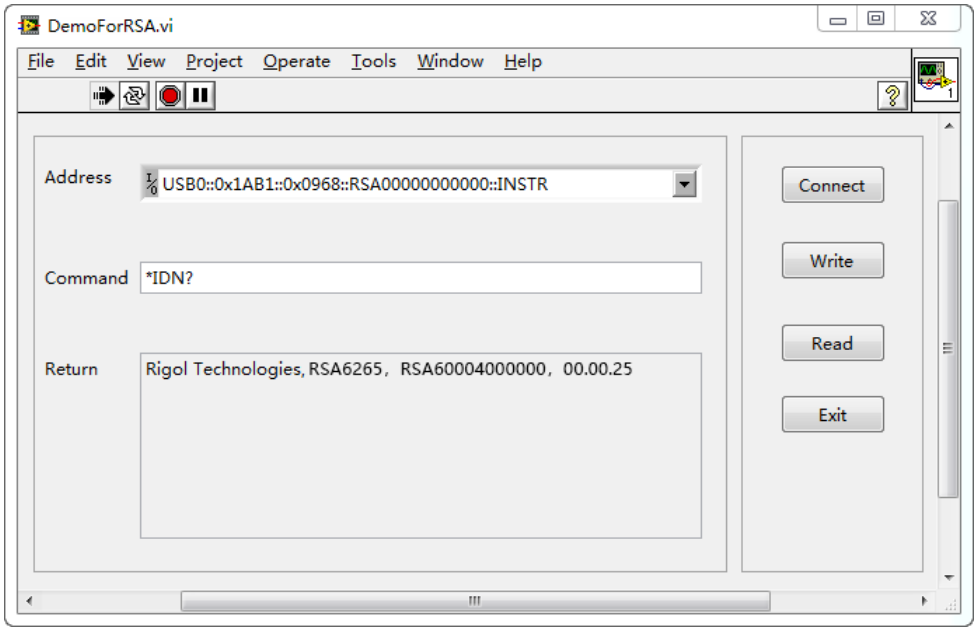





4) Exit




5) Run the program, and then the following interface is displayed below. Click the VISA resource name from the drop-down list under **Address**, and click **Connect** to connect the instrument. Then, input a command in the **Command** field. Click **Write** to write the command to the instrument. If the command is a query (e.g.*IDN?), click **Write** to write the command into the instrument, and then click **Read**. The return value is displayed in the **Return** field. Click **Exit** to exit the program.


DemoForRSA.vi
File Edit View Project Operate Tools Window Help
Address
USB0::0x1AB1::0x0968::RSA00000000000::INSTR
Command
*IDN?
Return
Rigol Technologies,RSA6265, RSA60004000000, 00.00.25
Connect
Write
Read
Exit

# Linux Programming Example

This section illustrates how to program and control the spectrum analyzer to realize the common functions in Linux operating system.

## Programming Preparations

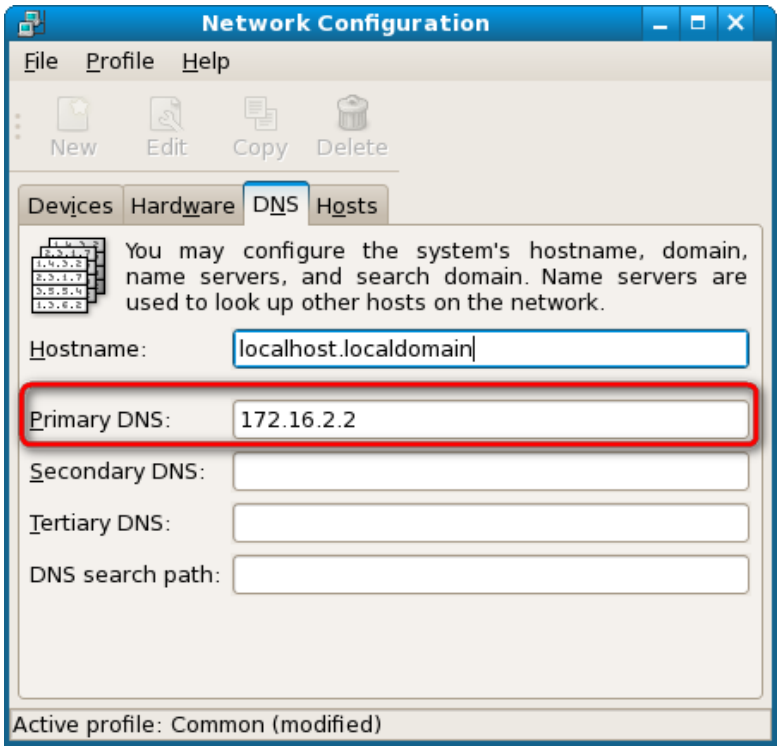
1. Programming environment:
   Operating system: Fedora 8 (Linux-2.6.23)
   GCC version: gcc-4.1.2

2. Install the VISA library. First, check whether your PC has installed NI's VISA library. If not, download it from NI website (http://www.ni.com/visa/). The installation procedures are as follows:
   Download the VISA library NI-VISA-4.4.0.ISO from the NI website.

   Create a new directory.
   #mkdir NI_VISA
   Mount the iso file.
   #mount -o loop -t iso9660 NI-VISA-4.4.0.iso NI_VISA
   Enter the NI_VISA directory to install
   #cd NI_VISA
   #./INSTALL
   Unmount the iso file.
   #umount NI_VISA
   After the installation is finished, the default installation path is /usr/local.

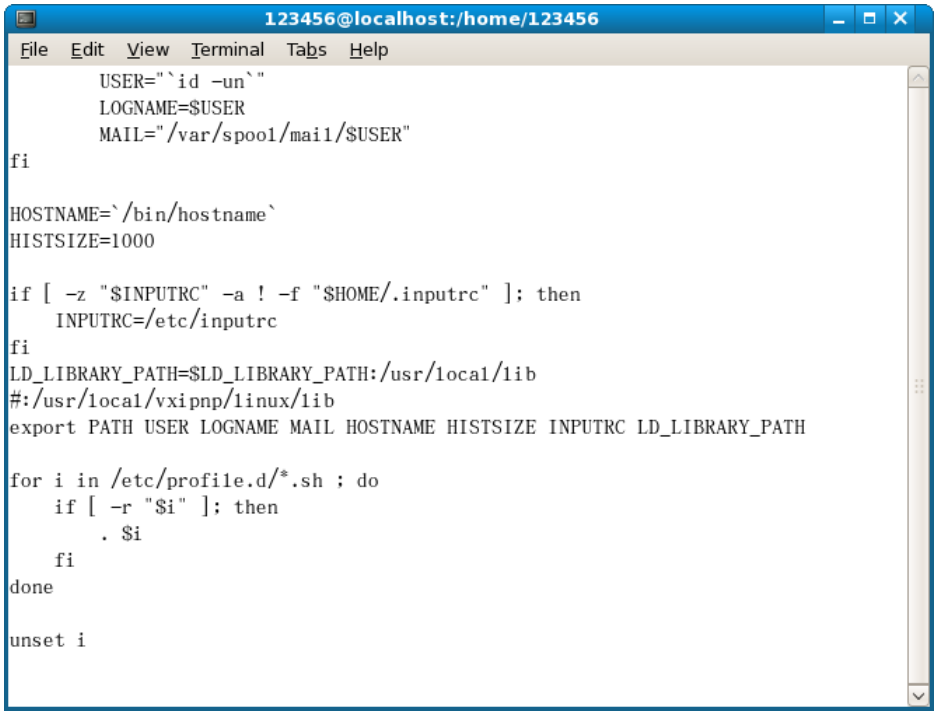
3. Connect spectrum analyzer to the PC via the LAN interface of the analyzer. Use the USB cable to connect the analyzer to the PC via the LAN interface on the rear panel of the analyzer. You can also use a network cable to connect the spectrum analyzer to the local area network where the PC resides.

   After the spectrum analyzer is connected to the PC properly, configure the network address for the spectrum analyzer to make its address to be within the same network segment where the PC resides. For example, if the network address and DNS setting configured for the PC are as shown in the figures below, then, the network address of the spectrum analyzer should be configured as follows.
   IP Address: 172.16.3.X*
   Default Gateway: 172.16.3.1
   Subnet Mask: 255.255.255.0
   DNS: 172.16.2.2
   Note*: X can be any value not used ranging from 2 to 254.

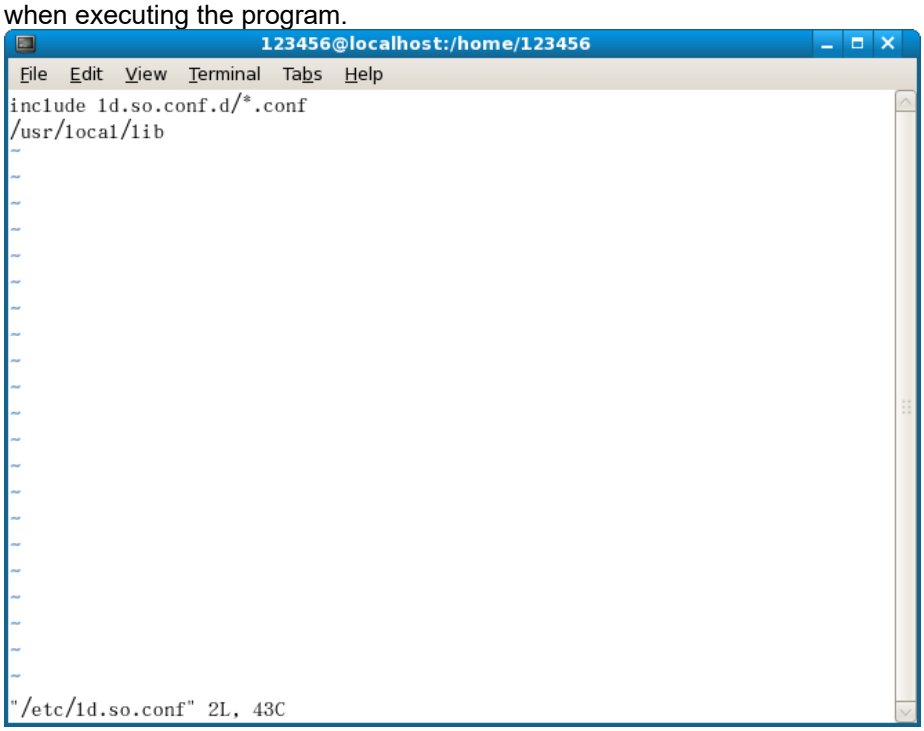




4. Use either of the following two methods to add the library location to the search path of the library, so that the program can load the installed library file automatically.

   **Method 1:** Specify the search path of the library in the environment variable LD_LIBRARY_PATH.
   **Operation Method:** Add the library file path /usr/local/lib to the LD_LIBRARY_PATH variable in the /etc/profile file, as shown in the figure below.




Method 2: Add the search path of the library in the /etc/ld.so.conf file.
Operation Method: #echo "/usr/local/lib" >> /etc/ld.so.conf, as shown in the figure below.
After setting the search path of the library in /etc/ld.so.conf, run the /sbin/ldconfig command to update /etc/ld.so.cache (this command should have the root permission) to ensure the location of the library when executing the program.




## Linux Programming Procedures

1. Edit the DemoForDSA.h header file and declare a class to encapsulate the operation and property of the instrument.
   #ifndef DEMO_FOR_RSA_H

```
#define DEMO_FOR_RSA_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
//#include <syswait.h>
using namespace std;

#define MAX_SEND_BUF_SIZE 50
#define MAX_REC_SIZE 300

class DemoForRSA
{
// Construction
public:
DemoForRSA();
bool InstrRead(string strAddr, string & pstrResult);
bool InstrWrite(string strAddr, string strContent);
bool ConnectInstr();

string m_strInstrAddr;
string m_strResult;
string m_strCommand;

};

void makeupper(string & instr);

#endif
```

2. Edit the DemoForDSA.cpp file to realize various operations of the instrument.

```
#include "visa.h"
#include "DemoForRSA.h"

DemoForRSA::DemoForRSA()
{
m_strInstrAddr = "";
m_strResult = "";
m_strCommand = "";
}

bool DemoForRSA::ConnectInstr()

{
ViUInt32 retCount;
ViStatus status;
ViSession defaultRM;
ViString expr = "?*";
ViPFindList findList = new unsigned long;
ViPUInt32 retcnt = new unsigned long;
string strSrc = "";
string strInstr = "";
ViChar instrDesc[1000];

unsigned long i = 0;
bool bFindRSA = false;
memset(instrDesc,0,1000);

//Turn on the VISA device
```

```
status = viOpenDefaultRM(&defaultRM);

if (status < VI_SUCCESS)
{
   cout<<"No VISA equipment!"<<endl;
   return false;
}

//Search for resources
status = viFindRsrc(defaultRM,expr,findList, retcnt, instrDesc);

for (i = 0;i < (*retcnt);i++)
{
   //Acquire the instrument name
   strSrc = instrDesc;

   InstrWrite(strSrc,"*IDN?");
   usleep(200);
   InstrRead(strSrc,strInstr);

   // If the RSA series is found, then exit
   makeupper(strInstr);
   if (strInstr.find("RSA",0) > 0)
   {
      bFindRSA = true;
      m_strInstrAddr = strSrc;
      break;
    }

   //Acquire the next device
   status = viFindNext(*findList,instrDesc);
}

if (bFindRSA == false)
{
    printf("RSA device not found!\n");
    return false;
}

    return true;
}

bool DemoForRSA::InstrWrite(string strAddr, string strContent) //Write operation
{
ViSession defaultRM,instr;
ViStatus status;
ViUInt32 retCount;
char * SendBuf = NULL;
char * SendAddr = NULL;
bool bWriteOK = false;
string str;
//Address conversion, convert the string type to char*
SendAddr = const_cast<char*>(strAddr.c_str());

//Address conversion, convert the string type to char*
SendBuf = const_cast<char*>(strContent.c_str());

//Turn on the specified device
status = viOpenDefaultRM(&defaultRM);
if (status < VI_SUCCESS)
```

```
{
   cout<<"No VISA equipment!"<<endl;
   return false;
}

status = viOpen(defaultRM, SendAddr, VI_NULL, VI_NULL, &instr);

//Write command to the device
status = viWrite(instr, (unsigned char *)SendBuf, strlen(SendBuf), &retCount);

//Turn off the device
status = viClose(instr);
status = viClose(defaultRM);
return bWriteOK;
}

bool DemoForRSA::InstrRead(string strAddr, string & pstrResult) //Read operation
{
ViSession defaultRM,instr;
ViStatus status;
ViUInt32 retCount;
char* SendAddr = NULL;
char * result = NULL;
bool bReadOK = false;
unsigned char RecBuf[MAX_REC_SIZE];
string str;
memset(RecBuf,0,MAX_REC_SIZE);

result=char*)malloc(MAX_REC_SIZE*sizeof(char));
memset(result,0,MAX_REC_SIZE);

//Address conversion, convert the string type to char*
SendAddr=const_cast<char*>(strAddr.c_str());

//Turn on the VISA device
status = viOpenDefaultRM(&defaultRM);
if (status < VI_SUCCESS)
{
    // Error Initializing VISA...exiting
    cout<<"No VISA equipment!"<<endl;
    return false;
}

//Turn on the specified device
status = viOpen(defaultRM, SendAddr, VI_NULL, VI_NULL, &instr);

//Read from the device
status = viRead(instr, RecBuf, MAX_REC_SIZE, &retCount);

//Turn off the device
status = viClose(instr);
status = viClose(defaultRM);
sprintf(result,"%s",RecBuf);
pstrResult = result;
free(result);
return bReadOK;
}

void makeupper( string &instr)
{
```

```
    string outstr = "";
    if(instr == "")
    {
       exit(0);
    }

    for(int i = 0;i < instr.length();i++)
    {
       instr[i] = toupper(instr[i]);
    }

}
```

3. Edit the function file mainloop.cpp to complete the flow control.

```
#include "DemoForRSA.h"

void menudisplay()
{
   cout<<"\t\t    Please operate the instrument:\n read write quit"<<endl;
}

int main()
{
   DemoForRSA demo;
   char temp[50];
   if(!demo.ConnectInstr())
   {
      cout<<"can not connect the equipment!"<<endl;
      return 0;
    }
   else
   {

      cout<<"\n connect equipment success!"<<endl;
      cout<<" the equipment address is :"<<demo.m_strInstrAddr<<endl;
    }

   while(1)
   {
      menudisplay();
      //cin>>demo.m_strCommand;
      cin.getline(temp,50);
      demo.m_strCommand=temp;
      if(demo.m_strCommand[0]='r' && demo.m_strCommand[1]='e'
         && demo.m_strCommand[2]='a' && demo.m_strCommand[3]='d')
      {
         //demo.InstrWrite(demo.m_strInstrAddr,"*IDN?");
         //demo.InstrRead(demo.m_strInstrAddr,demo.m_strResult);
         cout<<"read result:"<<demo.m_strResult<<endl;
         demo.m_strResult="";

       }

       else if (demo.m_strCommand[0]='w' && demo.m_strCommand[1]='r'
                  && demo.m_strCommand[2]='i' && demo.m_strCommand[3]='t' &&
                 demo.m_strCommand[4]='e')
       {
                if (demo.m_strInstrAddr="")
                {
```

```
                cout<<"Please connect the instrument!  \ n";
              }
             demo.InstrWrite(demo.m_strInstrAddr,demo.m_strCommand.substr(5,40));
             usleep(200);

           //Read operation
          demo.InstrRead(demo.m_strInstrAddr,demo.m_strResult);

       }

      else if (demo.m_strCommand[0] == 'q' && demo.m_strCommand[1] == 'u'
            && demo.m_strCommand[2] == 'i' && demo.m_strCommand[3] == 't')

      {
            break;
       }
      else if(demo.m_strCommand != "")
      {
            cout<<"Bad command!"<<endl;
       }
  }
  return 1;

}
```

4. makefile file

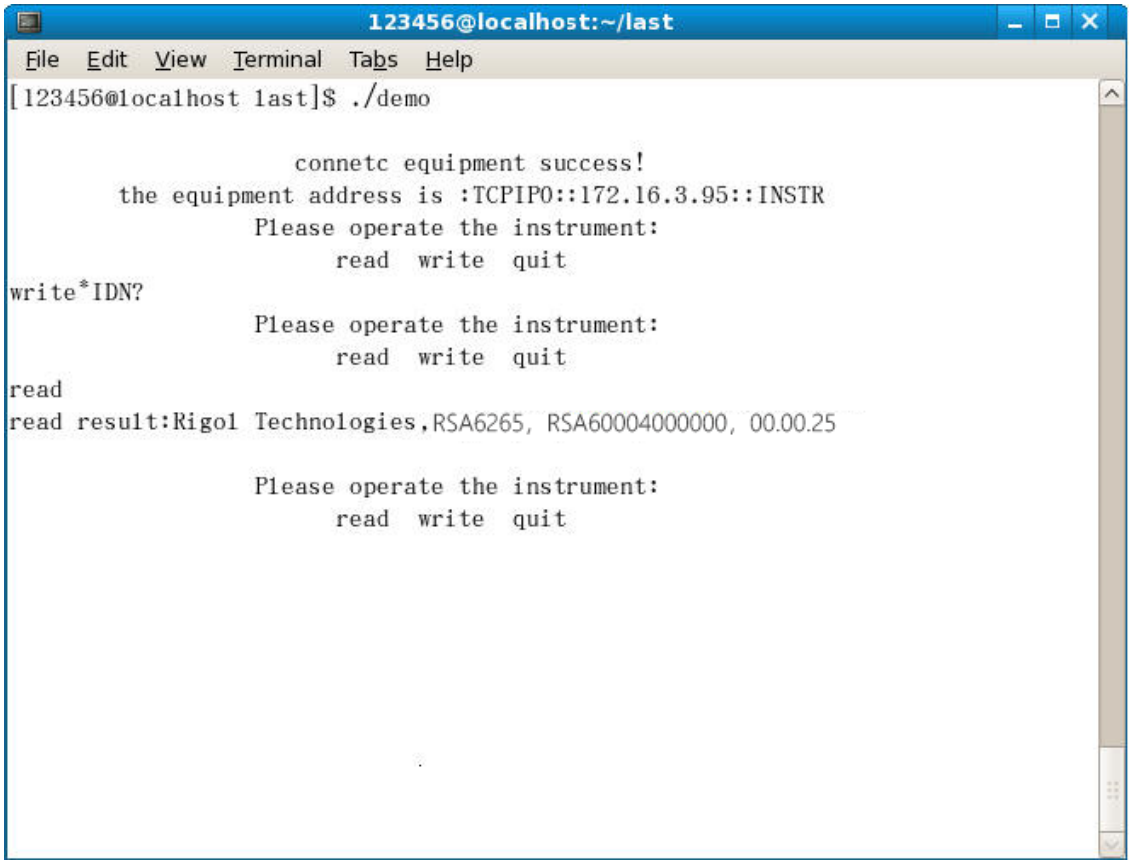
```
src = DemoForRSA.cpp mainloop.cpp DemoForRSA.h

obj = DemoForRSA.o mainloop.o
INCLUDE= -I/usr/local/vxipnp/linux/include
LIB= -lvisa -lc -lpthread
CC=
demo : $(obj)
$(CC) $(INCLUDE) $(LIB) -o demo $(obj)

mainloop.o : mainloop.cpp DemoForRSA.h
$(CC) -c $< -o $@
DemoForRSA.o: DemoForRSA.cpp DemoForRSA.h
$(CC) -c $< -o $@

.PHONY : clean
clean:
rm demo $(obj)
```

5. Run the results.
   1) #make
   2) ./demo
   3) When the program runs, the instrument is connected automatically. If no instrument is found, a prompt message "No VISA equipment!" is displayed, and the system exits the program. If the instrument is found and successfully connected, the following interface is displayed.
   4) Input write<command> (for example, write<*IDN?>) to write the command into the spectrum analyzer.
   5) Input read to read the return value, as shown in the figure below.


123456@localhost:~/last
File Edit View Terminal Tabs Help
[123456@localhost last]$ ./demo
connetc equipment success!
the equipment address is :TCPIP0::172.16.3.95::INSTR
Please operate the instrument:
read write quit
write*IDN?
Please operate the instrument:
read write quit
read
read result:Rigol Technologies,RSA6265, RSA60004000000, 00.00.25
Please operate the instrument:
read write quit

# Boost Smart World and Technology Innovation

Industrial Intelligent Manufacturing

Semiconductors

Education& Research


Communication

System Integration

New Energy

- Cellular-5G/WIFI
- UWB/RFID/ ZIGBEE
- Digital Bus/Ethernet
- Optical Communication

- Digital/Analog/RF Chip
- Memory and MCU Chip
- Third-Generation Semiconductor
- Solar Photovoltaic Cells

- New Energy Automobile
- PV/Inverter
- Power Test
- Automotive Electronics

*Provide Testing and Measuring Products and Solutions for Industry Customers*


**HEADQUARTER**

**RIGOL** TECHNOLOGIES CO., LTD.
No.8 Keling Road, New District,
Suzhou,JiangSu, P.R.China
Tel: +86-400620002
Email: info-cn@rigol.com

**EUROPE**

**RIGOL** TECHNOLOGIES EU GmbH
Friedrichshafener Str. 5
82205 Gilching
Germany
Tel: +49(0)8105-27292-21
Email: info-europe@rigol.com

**NORTH AMERICA**

**RIGOL** TECHNOLOGIES, USA INC.
10220 SW Nimbus Ave.
Suite K-7
Portland, OR 97223
Tel: +1-877-4-**RIGOL**-1
Email: sales@rigol.com

**JAPAN**

**RIGOL** JAPAN CO., LTD.
5F,3-45-6,Minamiotsuka, Toshima-Ku,
Tokyo,170-0005,Japan
Tel: +81-3-6262-8932
Fax: +81-3-6262-8933
Email: info.jp@rigol.com

**KOREA**

**RIGOL** KOREA CO,. LTD.
5F, 222, Gonghang-daero,
Gangseo-gu, Seoul, Republic of Korea
Tel: +82-2-6953-4466
Fax: +82-2-6953-4422
Email: info.kr@rigol.com

**For Assistance in Other Countries**

Email: info.int@rigol.com