Keysight B2980 Series Femto/Picoammeter Electrometer/High Resistance Meter

Programming Guide

KEYSIGHT TECHNOLOGIES

Notices

Manual Part Number

B2980-90020

Edition

Edition 1, September 2014

Keysight Technologies
1400 Fountaingrove Parkway
Santa Rosa, CA 95403

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Keysight disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Keysight shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Keysight and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Keysight Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Declaration of Conformity

To get the latest version of the declaration of conformity, go to http://www.keysight.com/go/conformity and type in the product number in the Search field.

Latest Information

To get the latest firmware/software/electronic manuals/specifications/support information, go to www.keysight.com and type in the product number in the Search field at the top of the page.

In This Manual

This manual provides the information for controlling Keysight Technologies B2980 by using an external computer, and consists of the following chapters.

- “Controlling Keysight B2980”
- “Programming Examples”

See *Keysight B2980 User's Guide* for information about the B2980 itself.

Refer to *Keysight B2980 SCPI Command Reference* for the SCPI messages and conventions, data output format, error code, and the details on Keysight B2980 SCPI commands.

Contents

Contents

1 Controlling Keysight B2980

KEYSIGHT TECHNOLOGIES

This chapter describes basic information to control Keysight B2980, and consists of the following sections.

- “Before Starting”
- “Controlling Various Functions”
- “Controlling the Measurement Function”
- “Controlling the Source Output”
- “Using the Math Function”
- “Performing the Limit Test”
- “Using the Trace Buffer”
- “Using Program Memory”

The following conventions are used in this document for expressing SCPI commands.

Convention	Description
Angle brackets < >	Items within angle brackets are parameter abbreviations. For example, <NR1> indicates a specific form of numerical data.
Vertical bar |	Vertical bars separate alternative parameters. For example, VOLT | CURR indicates that either “VOLT” or “CURR” can be used as a parameter.
Square brackets [ ]	Items within square brackets are optional. The representation [SENSe:]VOLTage means that SENSe: may be omitted.
Parentheses ( )	Items within parentheses are used in place of the usual parameter types to specify a channel list. The notation (@1:3) specifies a channel list that includes channels 1, 2, and 3. The notation (@1,3) specifies a channel list that includes only channels 1 and 3.
Braces { }	Braces indicate parameters that may be repeated zero or more times. It is used especially for representing arrays. The notation <A>{,<B>} shows that parameter “A” must be entered, while parameter “B” omitted or may be entered one or more times.

Before Starting

This section describes the information needed before starting programming.

- “Software Requirements”
- “Connecting to the Interface”
- “Starting the Instrument Control”

Software Requirements

Programming examples described in this manual use the following software. Install the software to your computer to execute the programming examples.

- Keysight IO Libraries Suite software
- Microsoft Visual Basic .NET software

Connecting to the Interface

Keysight B2980 supports GPIB, LAN, and USB interfaces. All three interfaces are live at power-on. Select the interface used for controlling the B2980. Connect your interface cable to the appropriate interface connector.

For the information on configuring the interfaces, see *Keysight B2980 Series User’s Guide*.

Starting the Instrument Control

The following program code is one of the simple program template for starting and ending the communication between the computer and the instrument. For using the code, the instrument address must be set to the `address` variable correctly.

```
Sub Main()
  Dim rm As Ivi.Visa.Interop.ResourceManager
  Dim ioObj As Ivi.Visa.Interop.FormattedIO488
  Dim address As String = "enter address of your instrument"

  rm = New Ivi.Visa.Interop.ResourceManager
  ioObj = New Ivi.Visa.Interop.FormattedIO488
  ioObj.IO = rm.Open(address)
      ' insert your code for instrument control
  ioObj.IO.Close()
End Sub
```

The `address` value depends on the interface as shown below.

- For using the GPIB interface

  The `address` value is the VISA GPIB Connect String displayed on the GPIB Configuration dialog box opened by pressing the System Menu > More > I/O > GPIB function keys.

  Example:

  ```
  address = "GPIB0::23::INSTR"
  ```

- For using the USB interface

  The `address` value is the VISA USB Connect String displayed on the USB Status dialog box opened by pressing the System Menu > More > I/O > USB function keys.

  Example:

  ```
  address = "USB0::2391::12345::XY00001234::0::INSTR"
  ```

- For using the LAN interface

  The `address` value is as follows.

  `address = "TCPIP0::`*xxx.yyy.zzz.aaa*`::5025::SOCKET"`

  Where, *xxx.yyy.zzz.aaa* is the IP Address displayed on the LAN Configuration dialog box opened by pressing the System Menu > More > I/O > LAN > Config function keys.

  Example:

  ```
  address = "TCPIP0::192.168.0.1::5025::SOCKET"
  ```

Controlling Various Functions

This section describes how to control various functions apart from the source output and measurement functions.

Setting the Power Frequency

Power line frequency is set by the :SYST:LFR command.

Example

```
ioObj.WriteString(":SYST:LFR 50") ’50 Hz
ioObj.WriteString(":SYST:LFR 60") ’60 Hz
ioObj.WriteString(":SYST:LFR:DET:AUTO") ’Automatic Detection
```

Resetting to the Initial Settings

The initial settings are applied by the *RST command

Example

```
ioObj.WriteString("*RST")
```

For the initial settings, see *SCPI Command Reference*.

Setting the Beeper

Beeper is enabled/disabled by the :SYST:BEEP:STAT command. And a beep sound of the specified frequency and duration is generated by the :SYST:BEEP command.

Example

```
ioObj.WriteString(":SYST:BEEP:STAT ON") 'Enables beep
ioObj.WriteString(":SYST:BEEP 200,1")   '200 Hz, 1 s
```

Setting the Date and Time

Date is set by the :SYST:DATE command. And time is set by the :SYST:TIME command.

Example

```
ioObj.WriteString(":SYST:DATE 2011,1,1")   'Y,M,D
ioObj.WriteString(":SYST:TIME 23,59,59")   'H,M,S
```

Performing the Self-Test

Self-test is performed by the *TST? command. The *TST? command also returns the execution result. Before performing the self-test, disconnect test leads and cables from the channel terminals.

Example

```
ioObj.WriteString("*TST?")
Dim d As String = ioObj.ReadString()
If d = 0 Then
  Console.WriteLine("PASS")
Else
  Console.WriteLine("FAIL")
End If
```

This example performs the self-test, and displays the test result, pass or fail.

Performing the Self-Calibration

Self-calibration is performed by the *CAL? command. The *CAL? command also returns the execution result. Before performing the self-calibration, disconnect test leads and cables from the channel terminals.

Example

```
ioObj.WriteString("*CAL?")
Dim d As String = ioObj.ReadString()
If d = 0 Then
  Console.WriteLine("PASS")
Else
  Console.WriteLine("FAIL")
End If
```

This example performs the self-calibration, and displays the result, pass or fail.

Setting the Operations at Power On

Operations at power-on are decided by the memory program specified by the :PROG:PON:COPY command. And the power-on program execution is enabled/ disabled by the :PROG:PON:RUN command. The specified program must be previously defined in the program memory.

Example

```
ioObj.WriteString(":PROG:PON:COPY ""program1""")
ioObj.WriteString(":PROG:PON:RUN ON")
```

This example sets *program1* to the power-on program and enables the function.

Reading an Error Message

Error message is read one by one by using the :SYST:ERR? command. This command reads and removes the top item in the error buffer, and returns the code and message.

Example

```
ioObj.WriteString(":SYST:ERR?")
Dim d As String = ioObj.ReadString()
Console.WriteLine(d)
```

If the error buffer is empty, the response is +0,“No error”.

Clearing the Error Buffer

Error buffer is cleared by the :SYST:ERR:ALL? command. This command reads and returns all items in the error buffer, and clears the buffer.

Example

```
ioObj.WriteString(":SYST:ERR:ALL?")
Dim d As String = ioObj.ReadString()
Console.WriteLine(d)
```

If the error buffer is empty, the response is +0,“No error”.

Reading Timestamp

Timestamp is read by the :SYST:TIME:TIM:COUN? command.

Example

```
ioObj.WriteString(":SYST:TIME:TIM:COUN?")
Dim d As String = ioObj.ReadString()
Console.WriteLine(d)
```

Clearing Timestamp

Timestamp is cleared by the :SYST:TIME:TIM:COUN:RES command.

Example

```
ioObj.WriteString(":SYST:TIME:TIM:COUN:RES")
```

Setting the Automatic Clear of Timestamp

Automatic clear of timestamp is enabled/disabled by the :SYST:TIME:TIM:COUN:RES:AUTO command. If this function is enabled, the timestamp is cleared when the initiate action occurs.

Example

```
ioObj.WriteString(":SYST:TIME:TIM:COUN:RES:AUTO ON")
```

Confirming the Firmware Revision

Instrument's (mainframe) identification and firmware revision are read by the *IDN? command.

Example

```
ioObj.WriteString("*IDN?")
Dim d As String = ioObj.ReadString()
Console.WriteLine(d)
```

The returned value will be as follows.

`Keysight Technologies,`*model*`,`*serial*`,`*revision*

model: mainframe model number

serial: mainframe serial number

revision: firmware revision number

Setting the Remote Display Mode

Front panel display under remote operation is enabled or disabled by the :DISP:ENAB command.

Example

```
ioObj.WriteString(":DISP:ENAB ON")
```

Making a Screen Dump

Screen dump of the front panel display is made by the :HCOP:SDUM commands.

Example

```
ioObj.WriteString(":DISP:ENAB ON")
ioObj.WriteString(":DISP:VIEW GRAP")
ioObj.WriteString(":HCOP:SDUM:FORM JPG")
ioObj.WriteString("*OPC?") : s = ioObj.ReadString()
ioObj.WriteString(":HCOP:SDUM:DATA?")

Dim data As Object
data = ioObj.ReadIEEEBlock(Ivi.Visa.Interop.IEEEBinaryType.Binary
Type_UI1, False, True)

Dim dataSize As Integer = data.Length
Dim dumpname As String =
"../../../execution result/screendump1.jpg"
Using stream As New FileStream(dumpname, FileMode.Create,
FileAccess.Write)
  stream.Write(data, 0, dataSize)
End Using
```

Performing a File Operation

File operation is effective for the USB memory connected to the front panel USB connector, and performed by the :MMEM commands. Error occurs if an USB memory is not connected.

Example

```
ioObj.WriteString(":MMEM:CAT?")    'Gets file catalog
s = ioObj.ReadString()

ioObj.WriteString(":MMEM:STOR:DATA ""test.dat""") 'Saves data
ioObj.WriteString("*OPC?") : s = ioObj.ReadString()

ioObj.WriteString(":MMEM:STOR:STAT ""test.sta""") 'Saves status
ioObj.WriteString("*OPC?") : s = ioObj.ReadString()

ioObj.WriteString(":MMEM:LOAD:STAT ""test.sta""")  'Loads status
```

Controlling the Measurement Function

This section describes how to control the measurement function of Keysight B2980.

- “Enabling the Current/Charge Measurement”
- “Setting the Measurement Mode”
- “Performing Spot Measurement”
- “Setting the Measurement Speed”
- “Setting the Measurement Range”
- “Setting the Measurement Trigger”
- “Performing Sweep Measurement”
- “Stopping Measurement”

Enabling the Current/Charge Measurement

The current/charge measurement is enabled by the :INP ON command.

Example

```
ioObj.WriteString(":INP ON")
```

Setting the Measurement Mode

Measurement mode is set by the :SENS:FUNC commands.

Example

```
ioObj.WriteString(":SENS:FUNC:ON ""VOLT"",""CURR"",""RES""")
ioObj.WriteString(":SENS:FUNC:ON ""VOLT"",""CHAR""")
```

Performing Spot Measurement

Spot measurement is performed by the :MEAS:<CURR|CHAR|VOLT|RES>? command or the :MEAS? command. See Figure 1-3 for the spot measurement.

Example

```
ioObj.WriteString(":MEAS:RES?")
```

```
ioObj.WriteString(":FORM:ELEM:SENS RES,STAT")
ioObj.WriteString(":MEAS?")
```

NOTE The :FORM:ELEM:SENS command is used to specify the data to obtain.

Setting the Measurement Speed

Measurement speed is set by the :SENS:<CURR|CHAR|VOLT|RES>:APER or :SENS:<CURR|CHAR|VOLT|RES>:NPLC command.

Example

```
ioObj.WriteString(":SENS:CURR:APER:AUTO ON") 'Automatic
ioObj.WriteString(":SENS:CURR:APER 1E-4")    '0.1 ms
ioObj.WriteString(":SENS:CURR:NPLC 1")       '1 Power Line Cycle
```

Setting the Measurement Range

Measurement range is set by the :SENS:<CURR|CHAR|VOLT|RES>:RANG command. And the auto range operation is enabled/disabled by the :SENS:<CURR|CHAR|VOLT|RES>:RANG:AUTO command. The lower limit for the auto range operation is set by the :SENS:<CURR|VOLT|RES>:RANG:AUTO:LLIM command. The upper limit for the auto range operation is set by the :SENS:<CURR|VOLT|RES>:RANG:AUTO:ULIM command. For charge measurement, the auto range operation is set by the :SENS:CHAR:RANG:AUTO:GRO command.

Example

```
ioObj.WriteString(":SENS:CURR:RANG:AUTO OFF")
ioObj.WriteString(":SENS:CURR:RANG 1E-12")         '1 pA fixed

ioObj.WriteString(":SENS:RES:RANG:AUTO ON")
ioObj.WriteString(":SENS:RES:RANG:AUTO:LLIM 1E+9") '1G ohm limit
ioObj.WriteString(":SENS:RES:RANG:AUTO:ULIM 1E+12")'1T ohm limit
```

Setting the Measurement Trigger

Measurement trigger is simply set by the :TRIG<:ACQ | [:ALL]>:SOUR, :TRIG<:ACQ | [:ALL]>:TIM, :TRIG<:ACQ | [:ALL]>:COUN, and :TRIG<:ACQ | [:ALL]>:DEL commands. See Figures 1-1, 1-2, and 1-3.

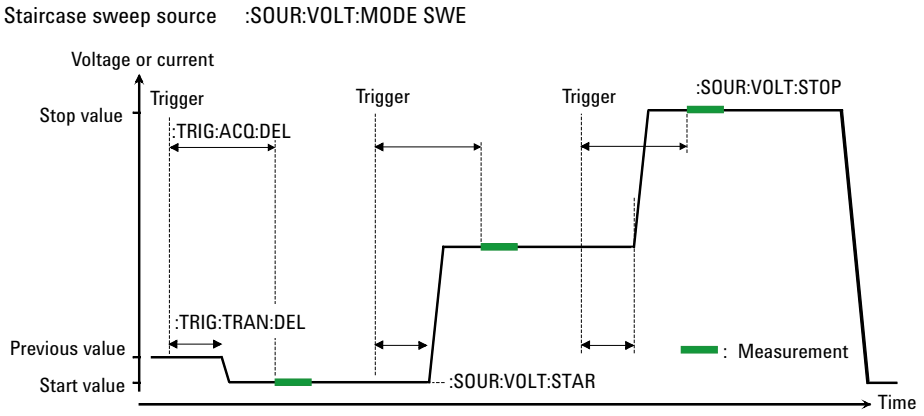
Example

```
ioObj.WriteString(":TRIG:SOUR TIM")
ioObj.WriteString(":TRIG:TIM 4E-3")       'Interval 4 ms
ioObj.WriteString(":TRIG:COUN 11")        '11 points
ioObj.WriteString(":TRIG:ACQ:DEL 2E-3")   'Meas delay 2 ms
```

NOTE

If you want to use arm trigger, use the :ARM<:ACQ | [:ALL]>:SOUR, :ARM<:ACQ | [:ALL]>:TIM, :ARM<:ACQ | [:ALL]>:COUN, and :ARM<:ACQ | [:ALL]>:DEL commands. For more details, see *SCPI Command Reference*.

Figure 1-1 **To Perform Staircase Sweep Output and Measurement**



Performing Sweep Measurement

Staircase sweep measurement is performed as shown below.

1. Set the staircase sweep source and the required source functions. For details, see “Controlling the Source Output” on page 1-13.
2. Set the required measurement functions. For details, see previous topics in this section.
3. Set the trigger condition. See “Setting the Source Output Trigger” on page 1-15 and “Setting the Measurement Trigger” on page 1-11.
4. Enable the channel. See “Enabling the Current/Charge Measurement” on page 1-10.

   The channel starts output set by the :SOUR:VOLT command.
5. Execute the :INIT command to start measurement.

For the programming example, see “Staircase Sweep Measurements” on page 2-9.

NOTE To specify the data to obtain, use the :FORM:ELEM:SENS command for the measurement data or the :FORM:ELEM:CALC command for the calculation data.

Stopping Measurement

Measurement is stopped by the :ABOR command.

Example

```
ioObj.WriteString(":ABOR")
```

Controlling the Source Output

This section describes how to control the source output of Keysight B2985 and B2987.

- "Enabling the Source Output"
- "Applying the DC Voltage"
- "Stopping the Source Output"
- "Setting the Output Range"
- "Setting the Sweep Operation"
- "Setting the Sweep Output"
- "Setting the Ranging Mode of the Sweep Source"
- "Setting the List Sweep Output"
- "Setting the Source Output Trigger"
- "Setting the Low Terminal State"
- "Specifying the Output-Off Status"

NOTE The string `:SOUR` in the command string described in this manual can be omitted. For example, `:SOUR:VOLT` can be `:VOLT`.

Enabling the Source Output

Source output is enabled by the :OUTP ON command.

Example

```
ioObj.WriteString(":OUTP ON")
```

Applying the DC Voltage

DC voltage is immediately applied by the :SOUR:VOLT command during the source output is enabled.

If you want to control the DC voltage output timing using a trigger, use the :SOUR:VOLT:TRIG command. See Figure 1-3.

Example

```
ioObj.WriteString(":SOUR:VOLT 20")      'Outputs 20 V immediately
```

```
ioObj.WriteString(":SOUR:VOLT:MODE FIX")
ioObj.WriteString(":SOUR:VOLT:TRIG -5") 'Outputs -5 V by a trigger
```

Stopping the Source Output

Source output is stopped and disabled by the :OUTP OFF command.

Example

```
ioObj.WriteString(":OUTP OFF")
```

Setting the Output Range

Output range is set by the :SOUR:VOLT:RANG command.

Example

```
ioObj.WriteString(":SOUR:VOLT:RANG -1000")     '-1000 V range
```

Setting the Sweep Operation

For the variety of sweep output operation, see Figure 1-2.

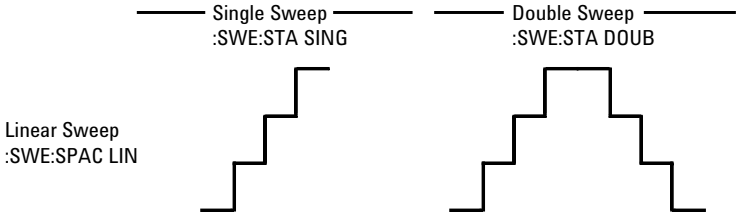
Sweep direction, upward or downward is set by the :SOUR:SWE:DIR command.

Sweep mode, single or double is set by the :SOUR:SWE:STA command.

Example

```
ioObj.WriteString(":SOUR:SWE:DIR DOWN")
ioObj.WriteString(":SOUR:SWE:STA DOUB")
```

Figure 1-2 Variety of Sweep Outputs



Setting the Sweep Output

Staircase sweep output is set by the :SOUR:VOLT:MODE SWE command, the :SOUR:VOLT:<POIN|STEP> or :SOUR:SWE:POIN command, and the :SOUR:VOLT:<STAR|STOP> or :SOUR:VOLT:<CENT|SPAN> command. See Figure 1-1.

Example

```
ioObj.WriteString(":SOUR:VOLT:MODE SWE")
ioObj.WriteString(":SOUR:VOLT:STAR 0")  'Start 0 V
ioObj.WriteString(":SOUR:VOLT:STOP 1")  'Stop 1 V
ioObj.WriteString(":SOUR:VOLT:POIN 11")  '11 points
```

NOTE

Outputting the sweep voltage

Execute the :OUTP ON command to start outputting the value set by the :SOUR:VOLT command.

Execute the :INIT to perform the specified sweep output and measurement.

Setting the Ranging Mode of the Sweep Source

Ranging mode of sweep source is set by the :SOUR:SWE:RANG command.

Example

```
ioObj.WriteString(":SOUR:SWE:RANG BEST")  'Covers all LIN steps
ioObj.WriteString(":SOUR:SWE:RANG FIX")   'Not change
```

Setting the List Sweep Output

List sweep output is set by the :SOUR:VOLT:MODE LIST command and the :SOUR:LIST:VOLT command

Example

```
ioObj.WriteString(":SOUR:VOLT:MODE LIST")
ioObj.WriteString(":SOUR:LIST:VOLT 0,2,4,6,8,10,0")
```

NOTE

Outputting the list sweep voltage/current

Execute the :OUTP ON command to start outputting the value set by the :SOUR:VOLT command.

Execute the :INIT to perform the specified list sweep output and measurement.

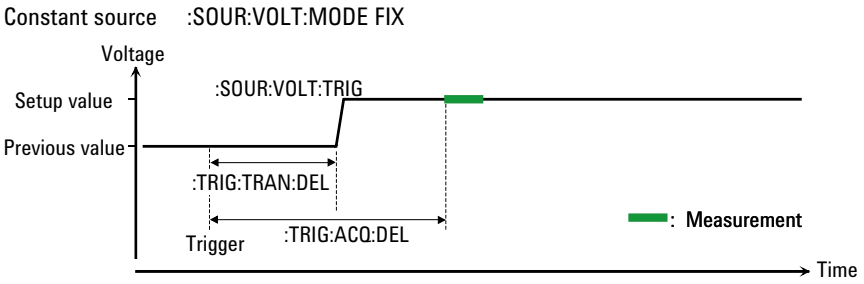
Setting the Source Output Trigger

Source output trigger is simply set by the :TRIG<:TRAN | [:ALL]>:SOUR, :TRIG<:TRAN | [:ALL]>:TIM, :TRIG<:TRAN | [:ALL]>:COUN, and :TRIG<:TRAN | [:ALL]>:DEL commands. See Figure 1-3.

Example

```
ioObj.WriteString(":TRIG:SOUR TIM")
ioObj.WriteString(":TRIG:TIM 4E-3")       'Interval 4 ms
ioObj.WriteString(":TRIG:COUN 11")        '11 points
ioObj.WriteString(":TRIG:TRAN:DEL 1E-3")  'Source delay 1 ms
```

Figure 1-3 **To Perform DC and Spot Measurement**



NOTE

If you want to use arm trigger, use the :ARM<:TRAN | [:ALL]>:SOUR, :ARM<:TRAN | [:ALL]>:TIM, :ARM<:TRAN | [:ALL]>:COUN, and :ARM<:TRAN | [:ALL]>:DEL commands. For more details, see *SCPI Command Reference*.

Setting the Low Terminal State

Low terminal state, ground or floating is set by the :OUTP:LOW command.

Example

```
ioObj.WriteString(":OUTP OFF")
ioObj.WriteString(":OUTP:LOW GRO")       'Ground
ioObj.WriteString(":OUTP ON")
```

Specifying the Output-Off Status

Output-off status is set by the :OUTP:OFF:MODE command.

Example

```
ioObj.WriteString(":OUTP:OFF:MODE ZERO")  'Zero volt
ioObj.WriteString(":OUTP:OFF:MODE HIZ")   'High impedance
ioObj.WriteString(":OUTP:OFF:MODE NORM")  'Normal
```

Using the Math Function

This section describes how to use the math function.

- “Defining a Mass Expression”
- “Deleting an User Defined Mass Expression”
- “Enabling or Disabling the Mass Function”
- “Reading Mass Result Data”

Defining a Mass Expression

Mass expression is defined by the :CALC:MATH[:EXPR] commands.

Example

```
ioObj.WriteString(":CALC:MATH:NAME ""DiffV""")
ioObj.WriteString(":CALC:MATH:DEF (SOUR-VOLT)")
ioObj.WriteString(":CALC:MATH:UNIT ""V""")
```

Deleting an User Defined Mass Expression

Mass expression is deleted by the :CALC:MATH[:EXPR]:DEL commands. The commands do not delete the predefined mass expression.

Example

```
ioObj.WriteString(":CALC:MATH:DEL ""DiffV""")   'Deletes DiffV
ioObj.WriteString(":CALC:MATH:DEL:ALL")         'Deletes all
```

Enabling or Disabling the Mass Function

Mass function is set by the :CALC:MATH:STAT command.

Example

```
ioObj.WriteString(":CALC:MATH:STAT ON")
```

Reading Mass Result Data

Mass result data is read by the :CALC:MATH:DATA? commands.

Example

```
ioObj.WriteString(":CALC:MATH:DATA:LAT?")   'Latest data
ioObj.WriteString(":CALC:MATH:DATA?")       'All data
```

NOTE To specify the data to obtain, use the :FORM:ELEM:CALC command.

Performing the Limit Test

Limit test is performed as shown below.

1. Set the required source condition. For details, see “Controlling the Source Output” on page 1-13.
2. Set the required measurement condition. For details, see “Controlling the Measurement Function” on page 1-10.
3. Set the composite limit test. See “Setting the Composite Limit Test” on page 1-19.
4. Set the individual limit tests. See “Setting Individual Limit Tests” on page 1-19.
5. Set the trigger condition. See “Setting the Source Output Trigger” on page 1-15 and “Setting the Measurement Trigger” on page 1-11.
6. Enable the channel. See “Enabling the Current/Charge Measurement” on page 1-10.
7. Execute the :INIT command to start limit test.

   Note that the DC source channel applies the value set by the :SOUR:VOLT:TRIG command when it is triggered, even if the channel is applying the value set by the :SOUR:VOLT command.
8. Read limit test result. See “Reading Limit Test Result” on page 1-19.

For the programming example, see “Pass/Fail Judgement and Math Function” on page 2-23.

Setting the Composite Limit Test

Composite limit test is set by the :CALC:CLIM commands and enabled by the :CALC:CLIM:STAT ON command. To perform a limit test, at least one individual limit test must be set and enabled.

Example

```
ioObj.WriteString(":CALC:CLIM:CLE")      'Clears result now
ioObj.WriteString(":CALC:CLIM:MODE GRAD")'Sets grading mode
ioObj.WriteString(":CALC:CLIM:UPD END")  'Sends result at the end
ioObj.WriteString(":CALC:CLIM:STAT ON")  'Composite limit test on
```

NOTE

If you want to use the GPIO port for sending a pass/fail bit pattern, use the :CALC:DIG commands to specify the output port. See *SCPI command reference*.

If you want to use the null offset function for cancelling the offset value from the measurement data automatically, use the :CALC:OFFS commands. See *SCPI command reference*.

Setting Individual Limit Tests

Individual limit test is set by the :CALC:LIM commands and the :CALC:FEED command. And each limit test is enabled by the :CALC:LIM:STAT ON command.

Example

```
ioObj.WriteString(":CALC:FEED CURR")      'Specifies feed data
ioObj.WriteString(":CALC:LIM:STAT ON")    'Limit test on
ioObj.WriteString(":CALC:LIM:FUNC LIM")   'Selects limit test
ioObj.WriteString(":CALC:LIM:UPP +3.5E-2")'Sets upper limit
ioObj.WriteString(":CALC:LIM:LOW +3E-2")  'Sets lower limit
```

Reading Limit Test Result

Limit test result is read by the :CALC:DATA? commands.

Fail status of the individual limit test is read by the :CALC:LIM:FAIL? command.

Example

```
ioObj.WriteString(":CALC:DATA:LAT?")       'Latest data
ioObj.WriteString(":CALC:DATA?")           'All data
ioObj.WriteString(":CALC:LIM:FAIL?")       '1:failed, 0:passed
```

NOTE

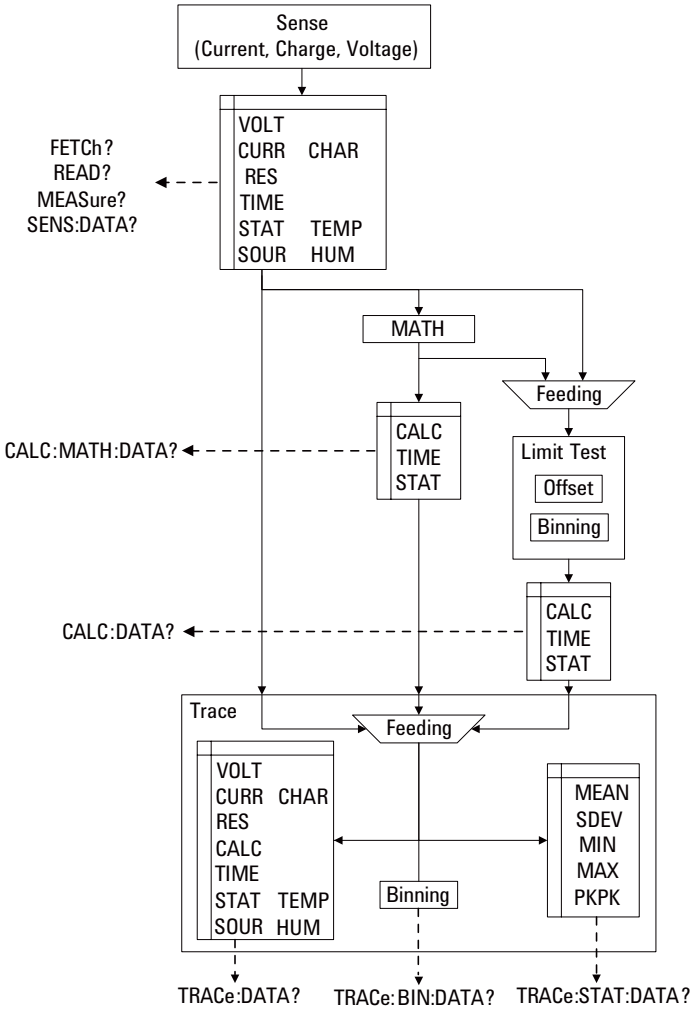
To specify the data to obtain, use the :FORM:ELEM:CALC command.

Using the Trace Buffer

This section describes how to use the trace buffer.

- “Setting the Trace Buffer”
- “Reading the Trace Data”

Figure 1-4 **Trace Buffer and Data Flow**



Setting the Trace Buffer

Trace buffer is set by the :TRAC commands.

Example

```
ioObj.WriteString(":TRAC:CLE")            'Clears trace buffer
ioObj.WriteString(":TRAC:POIN 1000")      'Sets buffer size
ioObj.WriteString(":TRAC:FEED SENS")      'Specifies data to feed
ioObj.WriteString(":TRAC:FEED:CONT NEXT") 'Enables write buffer
ioObj.WriteString(":TRAC:TST:FORM DELT")
```

NOTE

The :TRAC:TST:FORM command is used to specify the timestamp data format, delta (DELT) or absolute (ABS).

To specify the data to collect, use the :FORM:ELEM:SENS command for the measurement data or the :FORM:ELEM:CALC command for the calculation data.

Reading the Trace Data

All data in the trace buffer is read by the :TRAC:DATA? command.

Statistical data of the data stored in the trace buffer is read by the :TRAC:STAT:DATA? command. Previously, the type of the statistical data to read must be selected by the :TRAC:STAT:FORM command.

The :TRAC:STAT:FORM command selects one from the following statistical data.

- MEAN: Mean value
- SDEV: Standard deviation
- PKPK: Peak to peak value
- MIN: Minimum value
- MAX: Maximum value

Example

```
ioObj.WriteString(":TRAC:DATA?")          'Reads all data

ioObj.WriteString(":TRAC:STAT:FORM MEAN")
ioObj.WriteString(":TRAC:STAT:DATA?")     'Reads statistical data
```

Using Program Memory

This section describes how to use program memory.

- “Defining a Memory Program”
- “Deleting a Program”
- “Controlling the Program Operation”

Defining a Memory Program

Memory program is defined by the :PROG:NAME and :PROG:DEF commands.

Example

```
ioObj.WriteString(":PROG:NAME ""sample""")
ioObj.WriteString(":PROG:DEF #212:INP:STAT ON") 'Definite length

ioObj.WriteString(":PROG:NAME ""sample1""")
ioObj.WriteString(":PROG:DEF #0:INP:STAT ON") 'Indefinite length
```

Deleting a Program

Memory program is deleted by the :PROG:DEL commands.

Example

```
ioObj.WriteString(":PROG:DEL:ALL")          'Deletes all

ioObj.WriteString(":PROG:NAME ""sample1""")
ioObj.WriteString(":PROG:DEL")              'Deletes sample1
```

Controlling the Program Operation

Memory program is controlled by the :PROG:NAME command and the :PROG:EXEC or :PROG:STAT command. The :PROG[:SEL]:STAT command needs a parameter used to control the operation or change the status. The parameter must be RUN to change the status to running, PAUS to change it to paused, CONT to change it from paused to running, STOP to change it to stopped, or STEP to perform step execution.

Example

```
ioObj.WriteString(":PROG:NAME ""sample""")
ioObj.WriteString(":PROG:EXEC")
ioObj.WriteString("*OPC?") : s = ioObj.ReadString()

ioObj.WriteString(":PROG:NAME ""sample""")
ioObj.WriteString(":PROG:STAT RUN")
ioObj.WriteString("*OPC?") : s = ioObj.ReadString()
ioObj.WriteString(":PROG:STAT STOP")
```

2 Programming Examples

KEYSIGHT TECHNOLOGIES

This chapter provides the following sections which explain programming example.

- “Preparations”
- “Spot Measurements”
- “Staircase Sweep Measurements”
- “List Sweep Measurements”
- “Sampling Measurements”
- “Pass/Fail Judgement and Math Function”
- “Using Program Memory”
- “Reading Binary Data”
- “Reading Data during Measurement”

NOTE

About Example Program Code

Example programs described in this section have been written in the Microsoft Visual Basic .NET language. The examples are provided as a subprogram that can be run with the project template shown in Table 2-1. To run the program, insert the example subprogram or your subprogram instead of the B2900control subprogram in the template.

NOTE

To Start Program

If you create the measurement program by using the example code shown in Table 2-1, the program can be run by clicking the Run button on the Visual Basic main window.

Preparations

This section provides the basic information for programming of the automatic measurement using the Keysight B2980, Keysight IO Libraries, and Microsoft Visual Basic .NET.

- "To Create Your Project Template"
- "To Create Measurement Program"

NOTE To execute the example programs in this chapter, you need to install Keysight GPIB interface, Keysight IO Libraries Suite, and Microsoft Visual Basic .NET on your computer.

To Create Your Project Template

Before starting programming, create your project template, and keep it as your reference. It will remove the conventional task in the future programming. This section explains how to create a project template.

Step 1. Connect Keysight B2980 (e.g. GPIB address 23) to the computer via GPIB.

Step 2. Launch Visual Basic .NET and create a new project. The project type should be Console Application to simplify the programming.

Step 3. Add the following references to the project.

- VISA COM 3.0 (or later) Type Library
- Ivi.Visa.Interop
- System.IO

Step 4. Open a module (e.g. Module1.vb) in the project. And enter a program code as template. See Table 2-1 for example.

Step 5. Save the project as your template (e.g. \test\my_temp).

To Create Measurement Program

Create the measurement program as shown below. The following procedure needs your project template. If the procedure does not fit your programming environment, arrange it to suit your environment.

Step 1. Plan the automatic measurements. Then decide the following items:

- Measurement devices

  Discrete, packaged, on-wafer, and so on.

- Parameters/characteristics to be measured

  h_{FE}, Vth, sheet resistance, and so on.

- Measurement method

  Spot measurement, staircase sweep measurement, and so on.

Step 2. Make a copy of your project template (e.g. \test\my_temp to \test\dev_a\my_temp).

Step 3. Rename the copy (e.g. \test\dev_a\my_temp to \test\dev_a\spot1V).

Step 4. Launch Visual Basic .NET.

Step 5. Open the project (e.g. \test\dev_a\spot1V).

Step 6. Open the module that contains the template code as shown in Table 2-1. On the code window, complete the B2900control subprogram.

Step 7. Insert the code to display, store, or calculate data into the subprogram.

Step 8. Save the project (e.g. \test\dev_a\spot1V).

Table 2-1 Example Template Program Code

```
Module Module1

  Sub Main()                                                                    '1
    Dim rm As Ivi.Visa.Interop.ResourceManager
    Dim ioObj As Ivi.Visa.Interop.FormattedIO488
    Dim ifAddress As String = "23"
    Dim filename As String = ""
    Dim filedata As String = "Result: "
    Dim s As String = ""

    Try                                                                         '9
      rm = New Ivi.Visa.Interop.ResourceManager
      ioObj = New Ivi.Visa.Interop.FormattedIO488
      Try
        ioObj.IO = rm.Open("GPIB0::" + ifAddress + "::INSTR")
        ioObj.IO.Timeout = 60000
        ioObj.IO.TerminationCharacter = 10
        ioObj.IO.TerminationCharacterEnabled = True
      Catch ex As Exception
        Console.WriteLine("An error occurred: " + ex.Message)
      End Try

      B2900control(ioObj, s, filename)                                          '21
      Console.Write(filedata + s)
      MsgBox("Click OK to close the console window.", vbOKOnly, "")

      FileOpen(1, filename, OpenMode.Output, OpenAccess.Write,
OpenShare.LockReadWrite)                                                        '25
      Print(1, filedata + s)
      FileClose(1)

      ioObj.IO.Close()                                                          '29
      System.Runtime.InteropServices.Marshal.ReleaseComObject(ioObj)
      System.Runtime.InteropServices.Marshal.ReleaseComObject(rm)
    Catch ex As Exception
      Console.WriteLine("An error occurred: " + ex.Message)
    End Try
  End Sub

  Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As
String, ByRef filename As String)                                               '37
  filename = "../../../execution result/resultdata1.txt"
  End Sub

End Module
```

Line	Description
1 to 8	Beginning of the Main subprogram. And defines the variables used in this program.
9 to 20	Establishes the connection with the B2980 specified by the GPIB address ifAddress=23 on the interface GPIB0.

```
Module Module1

  Sub Main()                                                                    '1
    Dim rm As Ivi.Visa.Interop.ResourceManager
    Dim ioObj As Ivi.Visa.Interop.FormattedIO488
    Dim ifAddress As String = "23"
    Dim filename As String = ""
    Dim filedata As String = "Result: "
    Dim s As String = ""

    Try                                                                         '9
      rm = New Ivi.Visa.Interop.ResourceManager
      ioObj = New Ivi.Visa.Interop.FormattedIO488
      Try
        ioObj.IO = rm.Open("GPIB0::" + ifAddress + "::INSTR")
        ioObj.IO.Timeout = 60000
        ioObj.IO.TerminationCharacter = 10
        ioObj.IO.TerminationCharacterEnabled = True
      Catch ex As Exception
        Console.WriteLine("An error occurred: " + ex.Message)
      End Try

      B2900control(ioObj, s, filename)                                          '21
      Console.Write(filedata + s)
      MsgBox("Click OK to close the console window.", vbOKOnly, "")

      FileOpen(1, filename, OpenMode.Output, OpenAccess.Write,
OpenShare.LockReadWrite)                                                        '25
      Print(1, filedata + s)
      FileClose(1)

      ioObj.IO.Close()                                                          '29
      System.Runtime.InteropServices.Marshal.ReleaseComObject(ioObj)
      System.Runtime.InteropServices.Marshal.ReleaseComObject(rm)
    Catch ex As Exception
      Console.WriteLine("An error occurred: " + ex.Message)
    End Try
  End Sub

  Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As
String, ByRef filename As String)                                               '37
  filename = "../../../execution result/resultdata1.txt"
  End Sub

End Module
```

Line	Description
21 to 23	Calls the B2900control subprogram. And displays the result data in a console window.
25 to 27	Saves the result data to a file specified by filename.
29 to 35	Breaks the connection with the B2980.
37 to 39	B2900control subprogram. Measurement program code should be entered here.

Spot Measurements

A program example of spot measurements is shown in Table 2-2. This example is used to apply voltage and measure current.

Spot measurements can be performed by using the following commands.

Function	Command
Selects source function	:SOUR:FUNC:MODE VOLT
Sets source output range	:SOUR:VOLT:RANG *value*
Sets source output value	:SOUR:VOLT *value*
Sets measurement function	[:SENS:]FUNC "*func*"[, "*func*"[, "*func*"]]
Sets measurement range	[:SENS:]*func*:RANG:AUTO <ON | OFF>
	[:SENS:]*func*:RANG *value*
Sets aperture time in seconds or by using NPLC value	[:SENS:]*func*:APER *time*
	[:SENS:]*func*:NPLC *value*
Enables/disables Ammeter input	:INP <ON | OFF>
Enables/disables Voltage Source output	:OUTP <ON | OFF>
Initiates measurement and reads result data (latest data)	:MEAS?
	:MEAS:*func*?

Measurement Result Example

```
Result: +3.034160E-02
```

Table 2-2 Spot Measurement Example

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/FixedDc1.txt"                          ’2

  ioObj.WriteString("*RST") ’ Reset

  Try ’ Set voltage output to 0.1 V                                            ’6
    ioObj.WriteString(":sour:func:mode volt")
    ioObj.WriteString(":sour:volt 0.1")

    ’ Set auto-range current measurement
    ioObj.WriteString(":sens:func ""curr""")                                   ’11
    ioObj.WriteString(":sens:curr:rang:auto on")
    ioObj.WriteString(":sens:curr:nplc:auto off")
    ioObj.WriteString(":sens:curr:nplc 0.1")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on output switch
  ioObj.WriteString(":outp on")                                                ’21

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                 ’24

  Try ’ Initiate measurement and retrieve measurement result                   ’26
    ioObj.WriteString(":meas:curr?")
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
6 to 8	Sets the voltage source function. And sets the source value to 0.1 V.
11 to 14	Sets the current measurement function and the auto range measurement. And sets the aperture time to 0.1 PLC.
21	Enables the output and starts source output.
24	Enables the input.
26 to 28	Reads the measurement result data.

Staircase Sweep Measurements

A program example of staircase sweep measurements is shown in Table 2-3. This example is used to apply sweep voltage and measure current at each sweep step.

Staircase sweep measurements can be performed by using the following commands.

Function	Command
Selects source function	:SOUR:FUNC:MODE VOLT
Sets sweep output	:SOUR:VOLT:MODE SWE
Sets output range when starting sweep	:SOUR:VOLT:RANG *value*
Sets source output value	:SOUR:VOLT *value*
Sets sweep start or stop value	:SOUR:VOLT:<STAR | STOP> *value*
Sets sweep center or span value	:SOUR:VOLT:<CENT | SPAN> *value*
Sets sweep step value	:SOUR:VOLT:STEP *value*
Sets number of sweep steps	:SOUR:VOLT:POIN *value*
	:SOUR:SWE:POIN *value*
Selects sweep source ranging mode	:SOUR:SWE:RANG <BEST | FIX>
Selects sweep direction	:SOUR:SWE:DIR <UP | DOWN>
Selects sweep single or double	:SOUR:SWE:STA <SING | DOUB>
Sets measurement function	[:SENS:]FUNC "*func*"[, "*func*"[, "*func*"]]
Sets measurement range	[:SENS:]*func*:RANG:AUTO <ON | OFF>
	[:SENS:]*func*:RANG *value*
Sets aperture time in seconds or by using NPLC value	[:SENS:]*func*:APER *time*
	[:SENS:]*func*:NPLC *value*
Selects trigger source	:TRIG<:ACQ | :TRAN | [:ALL]>:SOUR *source*
Sets interval of timer trigger	:TRIG<:ACQ | :TRAN | [:ALL]>:TIM *time*

Function	Command
Sets trigger count	:TRIG<:ACQ | :TRAN | [:ALL]>:COUN *value*
Sets trigger delay time	:TRIG<:ACQ | :TRAN | [:ALL]>:DEL *time*
Enables/disables Ammeter input	:INP <ON | OFF>
Enables/disables Voltage Source output	:OUTP <ON | OFF>
Initiates specified action	:INIT<:ACQ | :TRAN | [:ALL]>
Reads result data (array data)	:FETC:ARR?
	:FETC:ARR:*type*?

func is VOLT for voltage measurement, CURR for current measurement, CHAR for charge measurement, or RES for resistance measurement.

source is AINT for the automatic trigger, BUS for the remote interface trigger command, TIM for the internal timer, INT*n* for a signal from the internal bus (*n*=1 or 2), EXT*m* for a signal from the GPIO pin *m* (*m*=1 to 7), or LAN for the LXI trigger.

type is VOLT for voltage data, CURR for current data, CHAR for charge date, RES for resistance data, SOUR for source output data, STAT for status data, or TIME for time data.

Measurement Result Example

```
Result: +1.842730E-07,+6.775910E-03,+1.519010E-02,+2.277660E-02,+
3.036230E-02
```

Table 2-3 Staircase Sweep Measurement Example

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/StaircaseSweep1.txt"                    '2

  ioObj.WriteString("*RST") ' Reset                                             '4

  Try ' Set voltage output from 0 V to 0.1 V, 5 steps
    ioObj.WriteString(":sour:func:mode volt")                                   '7
    ioObj.WriteString(":sour:volt:mode swe")
    ioObj.WriteString(":sour:volt:star 0")
    ioObj.WriteString(":sour:volt:stop 0.1")
    ioObj.WriteString(":sour:volt:poin 5")

    ' Set 2nA limited auto ranging current measurement
    ioObj.WriteString(":sens:func ""curr""")                                    '14
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ' Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               '18
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ' Generate 5 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                        '22
    ioObj.WriteString(":trig:coun 5")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ' Turn on output switch
  ioObj.WriteString(":outp on")                                                 '30

  ' Turn on input switch
  ioObj.WriteString(":inp on")                                                  '33

  ' Initiate transition and acquire
  ioObj.WriteString(":init")                                                    '36

  Try ' Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        '39
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
7 to 11	Sets the voltage sweep output function. And sets the sweep output from 0 to 0.1 V in 0.02 V step (5 points).

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/StaircaseSweep1.txt"                    ’2

  ioObj.WriteString("*RST") ’ Reset                                             ’4

  Try ’ Set voltage output from 0 V to 0.1 V, 5 steps
    ioObj.WriteString(":sour:func:mode volt")                                   ’7
    ioObj.WriteString(":sour:volt:mode swe")
    ioObj.WriteString(":sour:volt:star 0")
    ioObj.WriteString(":sour:volt:stop 0.1")
    ioObj.WriteString(":sour:volt:poin 5")

    ’ Set 2nA limited auto ranging current measurement
    ioObj.WriteString(":sens:func ""curr""")                                    ’14
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               ’18
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Generate 5 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                        ’22
    ioObj.WriteString(":trig:coun 5")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on output switch
  ioObj.WriteString(":outp on")                                                 ’30

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                  ’33

  ’ Initiate transition and acquire
  ioObj.WriteString(":init")                                                    ’36

  Try ’ Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        ’39
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
14 to 15	Sets the current auto ranging to 2nA limited auto ranging.
18 to 19	Sets the aperture time to 0.1 PLC
22 to 23	Sets the trigger source to AINT (automatic trigger). And sets the trigger count to 5 to perform a 5-step staircase sweep measurement.
30	Enables the output and starts source output (0 V with the default setting).

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/StaircaseSweep1.txt"                    ’2

  ioObj.WriteString("*RST") ’ Reset                                             ’4

  Try ’ Set voltage output from 0 V to 0.1 V, 5 steps
    ioObj.WriteString(":sour:func:mode volt")                                   ’7
    ioObj.WriteString(":sour:volt:mode swe")
    ioObj.WriteString(":sour:volt:star 0")
    ioObj.WriteString(":sour:volt:stop 0.1")
    ioObj.WriteString(":sour:volt:poin 5")

    ’ Set 2nA limited auto ranging current measurement
    ioObj.WriteString(":sens:func ""curr""")                                    ’14
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               ’18
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Generate 5 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                        ’22
    ioObj.WriteString(":trig:coun 5")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on output switch
  ioObj.WriteString(":outp on")                                                 ’30

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                  ’33

  ’ Initiate transition and acquire
  ioObj.WriteString(":init")                                                    ’36

  Try ’ Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        ’39
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
33	Enabled the input for current measurement.
36	Starts staircase sweep measurement.
39 to 40	Reads the measurement result data.

List Sweep Measurements

A program example of list sweep measurements is shown in Table 2-4. This example is used to apply sweep voltage and measure current at each sweep step.

List sweep measurements can be performed by using the following commands.

Function	Command
Selects source function	:SOUR:FUNC:MODE VOLT
Sets list sweep output	:SOUR:VOLT:MODE LIST
Sets output range when starting sweep	:SOUR:VOLT:RANG *value*
Sets source output value	:SOUR:VOLT *value*
Sets list sweep output values	:SOUR:LIST:VOLT *values*
Adds list sweep output values to the end of the present setting	:SOUR:LIST:VOLT:APP *values*
Specifies the list sweep start point	:SOUR:LIST:VOLT:STAR *start_index*
Asks the number of sweep points	:SOUR:LIST:VOLT:POIN?
Sets measurement function	[:SENS:]FUNC "*func*"[, "*func*"[, "*func*"]]
Sets measurement range	[:SENS:]*func*:RANG:AUTO <ON | OFF>
	[:SENS:]*func*:RANG *value*
Sets aperture time in seconds or by using NPLC value	[:SENS:]*func*:APER *time*
	[:SENS:]*func*:NPLC *value*
Selects trigger source	:TRIG<:ACQ | :TRAN | [:ALL]>:SOUR *source*
Sets interval of timer trigger	:TRIG<:ACQ | :TRAN | [:ALL]>:TIM *time*
Sets trigger count	:TRIG<:ACQ | :TRAN | [:ALL]>:COUN *value*
Sets trigger delay time	:TRIG<:ACQ | :TRAN | [:ALL]>:DEL *time*
Enables/disables Ammeter input	:INP <ON | OFF>

Function	Command
Enables/disables Voltage Source output	:OUTP <ON | OFF>
Initiates specified action	:INIT<:ACQ | :TRAN | [:ALL]>
Reads result data (array data)	:FETC:ARR?
	:FETC:ARR:*type*?

func is VOLT for voltage measurement, CURR for current measurement, CHAR for charge measurement, or RES for resistance measurement.

source is AINT for the automatic trigger, BUS for the remote interface trigger command, TIM for the internal timer, INT*n* for a signal from the internal bus (*n*=1 or 2), EXT*m* for a signal from the GPIO pin *m* (*m*=1 to 7), or LAN for the LXI trigger.

type is VOLT for voltage data, CURR for current data, RES for resistance data, SOUR for source output data, STAT for status data, or TIME for time data.

Measurement Result Example

```
Result: +9.121600E-03,+1.822320E-02,+3.036130E-02
```

Table 2-4 List Sweep Measurement Example

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/ListSweep1.txt"                          ’2

  ioObj.WriteString("*RST") ’ Reset                                              ’4

  Try ’ Set voltage output to 0.03, 0.06, and 0.1 V
    ioObj.WriteString(":sour:func:mode volt")                                    ’7
    ioObj.WriteString(":sour:volt:mode list")
    ioObj.WriteString(":sour:list:volt 0.03,0.06,0.1")

    ’ Set 2nA limited auto ranging
    ioObj.WriteString(":sens:func ""curr""")                                     ’12
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                                ’16
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Generate 3 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                         ’20
    ioObj.WriteString(":trig:coun 3")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on output switch
  ioObj.WriteString(":outp on")                                                  ’28

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                   ’31

  ’ Initiate transition and acquire
  ioObj.WriteString(":init")                                                     ’34

  Try ’ Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                         ’37
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
7 to 9	Sets the voltage list sweep output function. And sets the list sweep output 0.03 V, 0.06 V, and 0.1 V (3 points).

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/ListSweep1.txt"                          ’2

  ioObj.WriteString("*RST") ’ Reset                                              ’4

  Try ’ Set voltage output to 0.03, 0.06, and 0.1 V
    ioObj.WriteString(":sour:func:mode volt")                                    ’7
    ioObj.WriteString(":sour:volt:mode list")
    ioObj.WriteString(":sour:list:volt 0.03,0.06,0.1")

    ’ Set 2nA limited auto ranging
    ioObj.WriteString(":sens:func ""curr""")                                    ’12
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               ’16
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Generate 3 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                        ’20
    ioObj.WriteString(":trig:coun 3")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on output switch
  ioObj.WriteString(":outp on")                                                 ’28

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                  ’31

  ’ Initiate transition and acquire
  ioObj.WriteString(":init")                                                    ’34

  Try ’ Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        ’37
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
12 to 13	Sets the current auto ranging to 2nA limited auto ranging.
16 to 17	Sets the aperture time to 0.1 PLC
20 to 21	Sets the trigger source to AINT (automatic trigger). And sets the trigger count to 3to perform a 3-step list sweep measurement.

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/ListSweep1.txt"                          '2

  ioObj.WriteString("*RST") ' Reset                                              '4

  Try ' Set voltage output to 0.03, 0.06, and 0.1 V
    ioObj.WriteString(":sour:func:mode volt")                                    '7
    ioObj.WriteString(":sour:volt:mode list")
    ioObj.WriteString(":sour:list:volt 0.03,0.06,0.1")

    ' Set 2nA limited auto ranging
    ioObj.WriteString(":sens:func ""curr""")                                    '12
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")

    ' Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               '16
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ' Generate 3 triggers by automatic internal algorithm
    ioObj.WriteString(":trig:sour aint")                                        '20
    ioObj.WriteString(":trig:coun 3")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ' Turn on output switch
  ioObj.WriteString(":outp on")                                                 '28

  ' Turn on input switch
  ioObj.WriteString(":inp on")                                                  '31

  ' Initiate transition and acquire
  ioObj.WriteString(":init")                                                    '34

  Try ' Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        '37
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
28	Enables the channel. And starts source output (0 V with the default setting).
31	Enabled the input for current measurement.
33	Starts list sweep measurement.
37 to 38	Reads the measurement result data.

Sampling Measurements

A program example of sampling measurements is shown in Table 2-5. This example is used to apply constant voltage and measure current six times.

Sampling measurements can be performed by using the following commands.

Function	Command
Selects source function	:SOUR:FUNC:MODE VOLT
Sets output range when starting sweep	:SOUR:VOLT:RANG *value*
Sets source output value	:SOUR:VOLT *value*
Sets triggered source output value	:SOUR:VOLT:TRIG *value*
Sets measurement function	[:SENS:]FUNC "*func*"[, "*func*"[, "*func*"]]
Sets measurement range	[:SENS:]*func*:RANG:AUTO <ON | OFF>
	[:SENS:]*func*:RANG *value*
Sets aperture time in seconds or by using NPLC value	[:SENS:]*func*:APER *time*
	[:SENS:]*func*:NPLC *value*
Selects trigger source	:TRIG<:ACQ | :TRAN | [:ALL]>:SOUR *source*
Sets interval of timer trigger	:TRIG<:ACQ | :TRAN | [:ALL]>:TIM *time*
Sets trigger count	:TRIG<:ACQ | :TRAN | [:ALL]>:COUN *value*
Sets trigger delay time	:TRIG<:ACQ | :TRAN | [:ALL]>:DEL *time*
Enables/disables Ammeter input	:INP <ON | OFF>
Enables/disables Voltage Source output	:OUTP <ON | OFF>
Initiates specified action	:INIT<:ACQ | :TRAN | [:ALL]>
Reads result data (array data)	:FETC:ARR?
	:FETC:ARR:*type*?

func is VOLT for voltage measurement, CURR for current measurement, CHAR for charge measurement, or RES for resistance measurement.

source is AINT for the automatic trigger, BUS for the remote interface trigger command, TIM for the internal timer, INT*n* for a signal from the internal bus (*n*=1 or 2), EXT*m* for a signal from the GPIO pin *m* (*m*=1 to 7), or LAN for the LXI trigger.

type is VOLT for voltage data, CURR for current data, RES for resistance data, SOUR for source output data, STAT for status data, or TIME for time data.

Measurement Result Example

```
Result: +3.036250E-02,+3.036240E-02,+3.036190E-02,+3.036210E-02,+
3.036300E-02,+3.036270E-02
```

Table 2-5 Sampling Measurement Example

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/Sampling1.txt"                          ’2

  ioObj.WriteString("*RST") ’ Reset                                             ’4

  Try ’ Set fixed range current measurement
    ioObj.WriteString(":sens:func ""curr""")                                    ’7
    ioObj.WriteString(":sens:curr:rang:auto off")
    ioObj.WriteString(":sens:curr:rang 2e-9")

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               ’12
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Adjust trigger timing parameters
    ioObj.WriteString(":trig:acq:del 2.0e-3")                                   ’16

    ’ Generate 6 triggers in 4 ms period
    ioObj.WriteString(":trig:sour tim")                                         ’19
    ioObj.WriteString(":trig:tim 4e-3")
    ioObj.WriteString(":trig:coun 6")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ’ Turn on input switch
  ioObj.WriteString(":inp on")                                                  ’28

  ’ Initiate acquire
  ioObj.WriteString(":init:acq")                                                ’31

  Try ’ Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                        ’34
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
7 to 9	Sets the current measurement function and the fixed range measurement.
12 to 13	Sets the aperture time to 0.1 PLC.

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/Sampling1.txt"                          '2

  ioObj.WriteString("*RST") ' Reset                                             '4

  Try ' Set fixed range current measurement
    ioObj.WriteString(":sens:func ""curr""")                                    '7
    ioObj.WriteString(":sens:curr:rang:auto off")
    ioObj.WriteString(":sens:curr:rang 2e-9")

    ' Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                              '12
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ' Adjust trigger timing parameters
    ioObj.WriteString(":trig:acq:del 2.0e-3")                                  '16

    ' Generate 6 triggers in 4 ms period
    ioObj.WriteString(":trig:sour tim")                                        '19
    ioObj.WriteString(":trig:tim 4e-3")
    ioObj.WriteString(":trig:coun 6")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ' Turn on input switch
  ioObj.WriteString(":inp on")                                                 '28

  ' Initiate acquire
  ioObj.WriteString(":init:acq")                                               '31

  Try ' Retrieve measurement result
    ioObj.WriteString(":fetc:arr:curr?")                                       '34
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
16	Sets the acquire (measurement) delay time.
19 to 21	Sets the timer trigger source. And sets the timer interval to 4 ms, and the trigger count to 6 to perform a 6-point sampling measurement.
28	Enables the Ammeter input.
31	Starts sampling measurement.
34 to 35	Reads the measurement result data.

Pass/Fail Judgement and Math Function

A program example of pass/fail judgements is shown in Table 2-6. This example is used to perform power vs voltage sweep measurement and performs pass/fail judgement. Power is calculated by using the built-in math function.

Math function can be set by using the following commands.

Function	Command
Specifies math expression	:CALC:MATH:NAME "*name*"
Defines math expression	:CALC:MATH:DEF *definition*
Sets unit for the math expression	:CALC:MATH:UNIT
Enables/disables math function	:CALC:MATH:STAT <ON | OFF>

Pass/fail judgements can be performed by using the following commands.

Function	Command
Sets composite limit test mode to grading or sorting	:CALC:CLIM:MODE <GRAD | SORT>
Specifies test result output timing for the grading mode, end or immediate	:CALC:CLIM:UPD <END | IMM>
Enables/disables composite limit test	:CALC:CLIM:STAT <ON | OFF>
Specifies limit test feed data	:CALC:FEED *type*
Sets compliance check or limit test	:CALC:LIM:FUNC <COMP | LIM>
Sets limit test lower or upper limit	:CALC:LIM:<LOW | UPP> *value*
Sets compliance fail condition	:CALC:LIM:COMP:FAIL <OUT | IN>
Enables/disables limit test	:CALC:LIM:STAT <ON | OFF>
Asks limit test result pass or fail	:CALC:LIM:FAIL?

type is VOLT for voltage measurement data, CURR for current measurement data, CHAR for charge measurement data, RES for resistance measurement data, or MATH for math result data.

Table 2-6 Pass/Fail Judgement Example

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/Judgment1.txt"                          ’2

  ioObj.WriteString("*RST") ’ Reset                                             ’4

  Try ’ Set voltage sweep from 0.0 V to 10.0V, 21 linear steps
    ioObj.WriteString(":sour:func:mode volt")                                   ’7
    ioObj.WriteString(":sour:volt:mode swe")
    ioObj.WriteString(":sour:volt:star 0.0")
    ioObj.WriteString(":sour:volt:stop 10.0")
    ioObj.WriteString(":sour:volt:poin 21")
    ioObj.WriteString(":sour:swe:sta sing")

    ’ Set 2nA limited auto ranging
    ioObj.WriteString(":sens:curr:rang:auto:llim 2e-9")                         ’15

    ’ Set measurement speed to 0.1 PLC
    ioObj.WriteString(":sens:curr:nplc:auto off")                               ’18
    ioObj.WriteString(":sens:curr:nplc 0.1")

    ’ Enable predefined math expression power (voltage * current)
    ioObj.WriteString(":calc:math:name ""POWER""")                              ’22
    ioObj.WriteString(":calc:math:stat on")

    ’ Set limit test mode to "Grading"
    ioObj.WriteString(":calc:clim:mode grad")                                   ’26
    ioObj.WriteString(":calc:clim:upd end")
    ioObj.WriteString(":calc:clim:stat on")

    ’ Feed math result and judge it is in +/-5 mW or not
    ioObj.WriteString(":calc:feed math")                                        ’31
    ioObj.WriteString(":calc:lim1:stat on")
    ioObj.WriteString(":calc:lim1:func lim")
    ioObj.WriteString(":calc:lim1:low -5.0e-3")
    ioObj.WriteString(":calc:lim1:upp +5.0e-3")
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
7 to 12	Sets the voltage sweep output function. And sets the sweep output from 0 to 10 V, 21 points. Also sets the single sweep.
15	Sets the current auto ranging to 2nA limited auto ranging.
18 to 19	Sets the aperture time to 0.1 PLC.
22 to 23	Enables POWER math function.
26 to 28	Enables composite limit test of grading mode.
31 to 35	Enables limit test 1 which judges if POWER is between -5 mW and +5 mW.

```
    ' Set arm count to 1, trigger count to 21
    ioObj.WriteString(":arm:coun 1")                                    '37
    ioObj.WriteString(":trig:coun 21")

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try

  ' Turn on output switch
  ioObj.WriteString(":outp on")                                         '45

  ' Turn on input switch
  ioObj.WriteString(":inp on")                                          '48

  ' Initiate transition and acquire
  ioObj.WriteString(":init")                                            '51

  ' Wait for operation complete
  ioObj.WriteString("*OPC?")                                            '54
  Dim d As String = ioObj.ReadString()
  Console.Write("*OPC?: " + d)
  Console.WriteLine()

  Try ' Retrieve measurement result (not bin numbers)                   '59
    ioObj.WriteString(":calc:data?")
    s = ioObj.ReadString()

    ioObj.WriteString(":calc:lim1:fail?")                               '63
    d = ioObj.ReadString()
    If d = 0 Then
      Console.WriteLine("PASS")
    Else
      Console.WriteLine("FAIL")
    End If
    Console.WriteLine()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
37 to 38	Sets the arm count to 1, and the trigger count to 21 to perform a 21-step staircase sweep measurement.
45	Enables the output. And starts source output (0 V with the default setting).
48	Enables the input.
51	Starts staircase sweep measurement.
54 to 57	Waits for operation complete. And write “*OPC?: 1” on the console window when the operation is completed.
59 to 60	Reads the measurement result data.
63 to 70	Reads the judgement result. And write “PASS” or “FAIL” on the console window.

Measurement Result Example

```
Result:
-1.172735E-12,+5.042879E-08,+1.943762E-07,+4.422486E-07,+7.842360
E-07,+1.231724E-06,+1.758310E-06,+2.414372E-06,+3.137176E-06,+3.9
86594E-06,+4.897826E-06,+5.953960E-06,+7.072061E-06,+8.307558E-06
,+9.611092E-06,+1.106005E-05,+1.258392E-05,+1.420136E-05,+1.59113
6E-05,+1.773358E-05,+1.967062E-05
```

Using Program Memory

A program example for using program memory is shown in Table 2-7. This example is used to store a program in the program memory and execute it.

Program memory can be set and controlled by using the following commands.

Function	Command
Returns the names of all programs defined in the program memory	:PROG:CAT?
Specifies memory program	:PROG:NAME "*name*"
Defines memory program [a]	:PROG:DEF *program_code*
Adds program code to the end of the memory program [a]	:PROG:APP *program_code*
Sets a value to the variable specified by *n* [b]	:PROG:VAR*n* "*value*"
Executes memory program [a]	:PROG:EXEC
Changes status of memory program [a]	:PROG:STAT *operation*
Blocks other commands until the program execution status changes to Paused or Stopped [a]	:PROG:WAIT? *timeout_in_seconds*
Deletes a memory program [a]	:PROG:DEL
Deletes all memory programs	:PROG:DEL:ALL

a. This function is effective for the memory program previously specified by the :PROG:NAME command.
b. Variables can be used in the memory program. They must be expressed as %*n*% (*n*: integer. 1 to 100) in the memory program.

operation is RUN to change to the running status, PAUS to change to the paused status, CONT to change to the running status, STOP to change to the stopped status, or STEP to perform step execution.

Table 2-7 Example to Use Program Memory

```
Sub B2900control(ByVal ioObj As Ivi.Visa.Interop.FormattedIO488, ByRef s As String,
ByRef filename As String)
  filename = "../../../execution result/ProgramMemory1.txt"                    ’2

  ioObj.WriteString("*RST") ’ Reset                                            ’4

  Try ’ Build program                                                          ’6
    Dim program As String = ""
    program = ":sour:func:mode volt\n"
    program += ":sour:volt:mode swe\n"
    program += ":sour:volt:star -10.0\n"
    program += ":sour:volt:stop 10\n"
    program += ":sour:volt:poin 21\n"
    program += ":sens:func ""curr"",""volt""\n"
    program += ":sens:curr:rang:auto:llim 2e-9\n"
    program += ":sens:curr:nplc:auto off\n"
    program += ":sens:curr:nplc 0.1\n"
    program += ":arm:coun 1\n"
    program += ":trig:coun 21\n"
    program += ":outp 1\n"
    program += ":inp 1\n"
    program += ":init (@1)\n"

    ’ Get program length
    Dim sProgramLength As String = String.Format("{0:#}", program.Length)      ’24

    ioObj.WriteString(":prog:name ""sample""")                                 ’26
    ioObj.WriteString(":prog:def #" + sProgramLength.Length.ToString() +
sProgramLength + program)

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
```

Line	Description
2	Defines the file name used for saving the result data.
4	Resets the B2980.
6 to 21	Enters program code to the “program” variable. The program is for performing voltage source current/voltage measure sweep measurement from -10V to 10V, 21 points, with the aperture time 0.1 PLC.
24	Gets the program length (number of characters in the “program” variable).
26 to 27	Stores the program code to the program memory as the program name “sample”.

```
  ' Run program
  ioObj.WriteString(":prog:stat run")                                    '34

  ' Wait for operation complete
  ioObj.WriteString("*OPC?")                                             '37
  s = ioObj.ReadString()
  Console.Write("*OPC?: " + s)
  Console.WriteLine()

  Try ' Retrieve measurement result
    ioObj.WriteString(":fetch:arr:curr?")                                '43
    s = ioObj.ReadString()

  Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
End Sub
```

Line	Description
34	Executes the memory program.
37 to 40	Waits for operation complete. And write "*OPC?: 1" on the console window when the operation is completed.
43 to 44	Reads the measurement result data.

Measurement Result Example

```
Result: -7.480000E-05,+6.524500E-03,+1.311680E-02,+1.971080E-02,+
2.630190E-02,+3.289570E-02,+3.948990E-02,+4.607940E-02,+5.266580E
-02,+5.926410E-02,+6.585490E-02,+7.244700E-02,+7.903250E-02,+8.56
2770E-02,+9.221940E-02,+9.880730E-02,+1.053949E-01,+1.119873E-01,
+1.185849E-01,+1.251736E-01,+1.317632E-01
```

Reading Binary Data

A program example for reading binary data is shown in Table 2-8. This example is used to read data in the ASCII format and the 8-byte binary format.

This example code can be replaced with the code used to read data in a measurement program. For example, it can be used instead of the lines from 38 to 44 of the staircase sweep measurement program shown in Table 2-3.

Data output format can be controlled by using the following commands.

Function	Command
Sets the data output format	:FORM[:DATA] *format*
Sets byte order of binary data	:FORM:BORD *byte_order*

format is ASC for the ASCII data output format, REAL,32 for the IEEE-754 single precision format (4-byte data), or REAL,64 for the IEEE-754 double precision format (8-byte data).

byte_order is NORM for the normal byte order from byte 1 to byte 4 or 8, or SWAP for the reverse byte order from byte 4 or 8 to byte 1.

Measurement Result Example

```
Result: V (V), I (A), Time (sec), Status: -5.88E-05,2.85297E-06,0
.021938,21120.0222637,0.006749,0.030997,36480.0499995,0.0151897,0
.037096,41600.0750008,0.022776,0.041071,41600.0999998,0.0303624,0
.045048,4160
```

Table 2-8 Example to Read Binary Data

```
' Select measure data elements
ioObj.WriteString(":form:elem:sens volt,curr,time,stat")                    '2

' Retrieve measurement result & Output measurement result(Ascii format)
ioObj.WriteString(":form asc")                                              '5
ioObj.WriteString(":fetch:arr?")
Dim numOfElem As Integer = 4 'V, I, Time, Status
Dim data(numOfElem * trigCount - 1)
data = ioObj.ReadList(Ivi.Visa.Interop.IEEEASCIIType.ASCIIType_Any, ",")

Dim value As String = "V (V), I (A), Time (sec), Status: "
s = value
Console.WriteLine("ASCII format")
Console.WriteLine(value)
For i = LBound(data) To UBound(data)
  If (i + 1) Mod numOfElem = 0 Then
    Console.WriteLine(data(i).ToString())
    s = s + data(i).ToString()
  Else
    Console.Write(data(i).ToString() + ",")
    s = s + data(i).ToString() + ","
  End If
Next

Console.WriteLine()

' Retrieve measurement result & Output measurement result(Real64 format)
Console.WriteLine("REAL64 format")                                          '28
Console.WriteLine(value)
ioObj.WriteString(":form real,64")
ioObj.WriteString(":fetch:arr?")
Dim data64
data64 = ioObj.ReadIEEEBlock(Ivi.Visa.Interop.IEEEBinaryType.BinaryType_R8, False,
True)
For i = LBound(data64) To UBound(data64)
  If (i + 1) Mod numOfElem = 0 Then
    Console.WriteLine(data64(i).ToString())
  Else
    Console.Write(data64(i).ToString() + ",")
  End If
Next
Console.WriteLine()
```

Line	Description
2	Specifies the data to return. This example selects voltage measurement data, current measurement data, time data, and status data.
5 to 23	Reads the measurement result data in the ASCII format.
28 to 44	Reads the measurement result data in the REAL,64 format.

Reading Data during Measurement

A program example for reading data is shown in Table 2-9. This example is used to read data during measurement. This example returns data in ASCII format.

For example, this example code can be used instead of the lines from 4 to 39 of the sampling measurement program shown in Table 2-5.

Measurement data can be read by using the following command.

Function	Command
Read sense data	:SENS:DATA? [*offset*[, *size*]]

offset indicates the beginning of the data received *n* | CURRent | STARt (default).

size is the number of data to be received.

Measurement Result Example

```
Data(1) -1.282000E-12,+4.999999E+00
Data(2) -6.000000E-14,+1.500000E+01
Data(3) +7.760000E-13,+2.500000E+01
```

Table 2-9 Example to Read Data during Measurement

```
Dim buffer As String
ioObj.WriteString("*RST") ' Reset

Try ' Set fixed-range current measurement
  ioObj.WriteString(":sens:func ""curr""")                                  '5
  ioObj.WriteString(":sens:curr:rang:auto off")
  ioObj.WriteString(":sens:curr:rang 2e-9")

  ' Set measurement speed to 0.1 PLC
  ioObj.WriteString(":sens:curr:nplc:auto off")                             '10
  ioObj.WriteString(":sens:curr:nplc 0.1")

  ' Adjust trigger timing parameters
  ioObj.WriteString(":trig:acq:del 5.0")                                    '14

  ' Generate 3 triggers in 10 s period
  ioObj.WriteString(":trig:acq:sour tim")                                   '17
  ioObj.WriteString(":trig:acq:tim 10.0")
  ioObj.WriteString(":trig:acq:coun 3")

Catch ex As Exception
  Console.WriteLine("An error occurred: " + ex.Message)
End Try

' Turn on input switch
ioObj.WriteString(":inp on")                                                '26

' Initiate acquire trigger with timestamp reset
ioObj.WriteString(":syst:time:tim:coun:res:auto on")                        '29
ioObj.WriteString(":init:acq")                                              '30

' Query current and timestamp
ioObj.WriteString(":form:elem:sens CURR,TIME")                              '31

' Retrieving measurement result
For count As Integer = 1 To 3 Step 1
  Try
    ioObj.WriteString(":sens:data? curr,1")                                 '36
    buffer = ioObj.ReadString()
    Console.Write("Data(" + count.ToString() + ") " + buffer)
 Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
Next
```

Line	Description
5 to 7	Set 2 nA fixed ranging current measurement.
10 to 11	Set 0.1 PLC measurement speed.
17 to 19	Set 3 points measurement with 10 seconds period.
26	Turn on input switch for current measurement.

```
Dim buffer As String
ioObj.WriteString("*RST") ' Reset

Try ' Set fixed-range current measurement
  ioObj.WriteString(":sens:func ""curr""")                                  '5
  ioObj.WriteString(":sens:curr:rang:auto off")
  ioObj.WriteString(":sens:curr:rang 2e-9")

  ' Set measurement speed to 0.1 PLC
  ioObj.WriteString(":sens:curr:nplc:auto off")                             '10
  ioObj.WriteString(":sens:curr:nplc 0.1")

  ' Adjust trigger timing parameters
  ioObj.WriteString(":trig:acq:del 5.0")                                    '14

  ' Generate 3 triggers in 10 s period
  ioObj.WriteString(":trig:acq:sour tim")                                   '17
  ioObj.WriteString(":trig:acq:tim 10.0")
  ioObj.WriteString(":trig:acq:coun 3")

Catch ex As Exception
  Console.WriteLine("An error occurred: " + ex.Message)
End Try

' Turn on input switch
ioObj.WriteString(":inp on")                                                '26

' Initiate acquire trigger with timestamp reset
ioObj.WriteString(":syst:time:tim:coun:res:auto on")                        '29
ioObj.WriteString(":init:acq")                                              '30

' Query current and timestamp
ioObj.WriteString(":form:elem:sens CURR,TIME")                              '31

' Retrieving measurement result
For count As Integer = 1 To 3 Step 1
  Try
    ioObj.WriteString(":sens:data? curr,1")                                 '36
    buffer = ioObj.ReadString()
    Console.Write("Data(" + count.ToString() + ") " + buffer)
 Catch ex As Exception
    Console.WriteLine("An error occurred: " + ex.Message)
  End Try
Next
```

Line	Description
29	Set timestamp to be reset automatically by INITiate command.
30	Initiate acquire trigger system.
31	Set data query format to current and timestamp.
36	Query one measurement result without waiting operation complete.

Edition 1, September 2014

B2980-90020
www.keysight.com

KEYSIGHT
TECHNOLOGIES