Programming Guide

Digital Oscilloscopes Series

RC01020-E01C

SIGLENT TECHNOLOGIES CO., LTD
Catalogue

Programming Overview .......................................................... 4

Build communication.............................................................. 4
  Install NI-VISA..................................................................... 4
  Connect the instrument ......................................................... 8

How To Remote Control............................................................ 9
  a. User-defined Programming ............................................. 9
  b. Send SCPI Commands via NI-VISA .................................. 9

About these Commands & Queries .......................................... 10

  How are they listed?............................................................. 10
  How are they described?....................................................... 10
  Where can they be used? .................................................... 10

Command Notation.................................................................... 12

Table of Commands & Queries .............................................. 13

Programming Examples......................................................... 149

  Example of VC++ ............................................................... 150
  Example of VB .................................................................... 159
  Example of MATLAB ......................................................... 167
  Example of LabVIEW .......................................................... 170
  Example of C# .................................................................... 173
Using Sockets Examples ........................................ 176
Example of C .................................................................. 176
Index ............................................................................. 178
Programming Overview

This chapter introduces how to execute remote communications between a SIGLENT digital oscilloscope and the computer. It also introduces how to establish a remote control link over a communication bus.

Build communication

Install NI-VISA

Before programming, you need to install National Instruments NI-VISA library, which you can download from the National Instruments web-site. Currently, NI-VISA is packaged in two versions: a full version and a Run-Time Engine version. The full version includes the NI device drivers and a tool named NI MAX that is a user interface to control the device. The Run-Time Engine is much smaller than the full version and only includes NI device driver.

For example, you can get the NI-VISA 5.4 full version from: http://www.ni.com/download/ni-visa-5.4/4230/en/.

You also can download NI-VISA Run-Time Engine 5.4 to your PC and install it as the default selection. Its installation process is similar with the full version.

After you downloaded the file you can follow the steps below to install it:
a. Double click the visa540_full.exe, dialog shown as below:

![WinZip Self-Extractor](image)

b. Click Unzip, the installation process will automatically launch after unzipping files. If your computer needs to install .NET Framework 4, its Setup process will auto start.

c. The NI-VISA installing dialog is shown above. Click Next to
start the installation process.

Set the install path, default path is “C:\Program Files\National Instruments\”, you can change it. Click Next, dialog shown as above.

d. Click Next twice, in the License Agreement dialog, select the
“I accept the above 2 License Agreement(s).”, and click Next, dialog shown as below:

![Digital Oscilloscopes Series](image)

---

**e. Click Next to run installation.**

![Digital Oscilloscopes Series](image)

---

Installation complete! You might be prompted to reboot your machine.

The best way to explore the new features is to use the VISA Interactive Control to open sessions to the new resource types and look at available operations, events, and attributes.

You can use Measurement and Automation Explorer to configure the settings for NI-VISA.

---

Now the installation is complete, reboot your PC.
Connect the instrument

Depending on your specific model your oscilloscope may be able to communicate with a PC through the USB or LAN interface. This manual takes the USB as an example. (For instructions to communicate with a PC through the LAN interface see the User Manual.)

a. Connect the USB Device interface at the rear panel of the oscilloscope and the USB Host interface of the PC using a USB cable. Assuming your PC is already turned on, turn on your oscilloscope and your PC will display the “Device Setup” screen as it automatically installs the device driver as shown below.

![Device Setup]

b. Wait for the installation to complete and then proceed to the next step.
How To Remote Control

a. User-defined Programming

Users can use SCPI commands to program and control the digital oscilloscope. For details, refer to the introductions in "Programming Examples".

b. Send SCPI Commands via NI-VISA

You can control the oscilloscope remotely by sending SCPI commands via NI-VISA software.
About these Commands & Queries

This section lists describes the remote control commands and queries recognized by the instrument. All commands and queries can be executed in either local or remote state.

The description for each command or query, with syntax and other information, begins on a new page. The name (header) is given in both long and short form at the top of the page, and the subject is indicated as a command or query or both. Queries perform actions such as obtaining information, and are recognized by the question mark (?) following the header.

How are they listed?

The descriptions are listed in alphabetical order according to their long form. Thus the description of ATTENUATION, whose short form is ATTN, is listed before that of AUTO SETUP, whose short form is ASET.

How are they described?

In the descriptions themselves, a brief explanation of the function performed is given. This is followed by a presentation of the formal syntax, with the header given in Upper-and-Lower-Case characters and the short form derived from it in ALL UPPER-CASE characters. Where applicable, the syntax of the query is given with the format of its response.

Where can they be used?

The commands and queries listed here can be used for all Siglent’s Digital Oscilloscope Series digital instruments.
Digital Oscilloscopes Series

<table>
<thead>
<tr>
<th>Applicable to the following models</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDS1000CML/CML+</td>
</tr>
<tr>
<td>SDS1000DL/DL+</td>
</tr>
<tr>
<td>SDS1000CNL/CNL+</td>
</tr>
<tr>
<td>SDS1000/1000X/1000X-S/1000X+/1000X-E</td>
</tr>
<tr>
<td>SDS2000/SDS2000X</td>
</tr>
</tbody>
</table>

Certain commands are only applicable to SPO oscilloscopes models and are described accordingly.

SPO oscilloscopes models are in the table below.

<table>
<thead>
<tr>
<th>SPO models</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDS1000X/1000X+/SDS1000X-E</td>
</tr>
<tr>
<td>SDS2000/2000X</td>
</tr>
</tbody>
</table>

What is an SPO model?
SPO model uses Siglent-innovated waveform acquisition and graphics processing engine which supports high capture rate, multi-level intensity grading and color temperature display, with deep memory storage and the use of new digital trigger technology supports rich trigger types and precise trigger. All of these technologies are collectively known as SPO (Super Phosphor Oscilloscope) technology.
Command Notation

The following notation is used in the commands:

< > Angular brackets enclose words that are used as placeholders, of which there are two types: the header path and the data parameter of a command.

: = A colon followed by an equals sign separates a placeholder from the description of the type and range of values that may be used in a command instead of the placeholder.

{} Braces enclose a list of choices, one of which one must be made.

[ ] Square brackets enclose optional items.

… An ellipsis indicates that the items both to its left and right may be repeated a number of times.

As an example, consider the syntax notation for the command to set the vertical input sensitivity:

<channel>:VOLT_DIV <v_gain>
<channel> : = {C1, C2, C3, C4}
<v_gain>: = 2 mV to 10 V

The first line shows the formal appearance of the command, with <channel> denoting the placeholder for the header path and <v_gain> the placeholder for the data parameter specifying the desired vertical gain value. The second line indicates that one of four channels must be chosen for the header path. And the third explains that the actual vertical gain can be set to any value between 2 mV and 10 V.
# Table of Commands & Queries

<table>
<thead>
<tr>
<th>Short Form</th>
<th>Long Form</th>
<th>Subsystem</th>
<th>What the Command or Query Does</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACQW</td>
<td>ACQUIRE WAY</td>
<td>ACQUISITION</td>
<td>Specifies the acquisition mode.</td>
</tr>
<tr>
<td>ALST?</td>
<td>ALL STATUS?</td>
<td>STATUS</td>
<td>Reads and clears the contents of all status registers.</td>
</tr>
<tr>
<td>ARM</td>
<td>ARM ACQUISITION</td>
<td>ACQUISITION</td>
<td>Changes acquisition state from “stopped” to “single”.</td>
</tr>
<tr>
<td>ATTN</td>
<td>ATTENUATION</td>
<td>CHANNEL</td>
<td>Selects the vertical attenuation factor of the probe.</td>
</tr>
<tr>
<td>ACAL</td>
<td>AUTO CALIBRATE</td>
<td>SYSTEM</td>
<td>Enables or disables automatic calibration.</td>
</tr>
<tr>
<td>ASET</td>
<td>AUTO SETUP</td>
<td>ACQUISITION</td>
<td>Adjusts vertical, time base and trigger parameters.</td>
</tr>
<tr>
<td>AUTTS</td>
<td>AUTO TYPESET</td>
<td>ACQUISITION</td>
<td>Selects the display type of automatic setup.</td>
</tr>
<tr>
<td>AVGA</td>
<td>AVERAGE ACQUIRE</td>
<td>ACQUISITION</td>
<td>Selects the average times of average acquisition.</td>
</tr>
<tr>
<td>BWL</td>
<td>BANDWIDTH LIMIT</td>
<td>CHANNEL</td>
<td>Enables/disables the bandwidth-limiting low-pass filter.</td>
</tr>
<tr>
<td>BUZZ</td>
<td>BUZZER</td>
<td>SYSTEM</td>
<td>Controls the built-in piezo-electric buzzer.</td>
</tr>
<tr>
<td>*CAL?</td>
<td>*CAL?</td>
<td>SYSTEM</td>
<td>Performs complete internal calibration of the instrument.</td>
</tr>
<tr>
<td>CHDR</td>
<td>COMM HEADER</td>
<td>SYSTEM</td>
<td>Controls formatting of query responses.</td>
</tr>
<tr>
<td>*CLS</td>
<td>*CLS</td>
<td>STATUS</td>
<td>Clears all status data registers.</td>
</tr>
<tr>
<td>CMR?</td>
<td>CMR?</td>
<td>STATUS</td>
<td>Reads and clears the Command error Register (CMR).</td>
</tr>
<tr>
<td>CONET</td>
<td>COMM NET</td>
<td>SYSTEM</td>
<td>Specifies network addresses of scope and printers.</td>
</tr>
<tr>
<td>CPL</td>
<td>COUPLING</td>
<td>CHANNEL</td>
<td>Selects the specified input channel’s coupling mode.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CRMS</strong></td>
<td>CURSOR_MEASURE</td>
<td>CURSOR</td>
<td>Specifies the type of cursor/parameter measurement.</td>
</tr>
<tr>
<td><strong>CRST</strong></td>
<td>CURSOR_SET?</td>
<td>CURSOR</td>
<td>Allows positioning of any one of eight cursors.</td>
</tr>
<tr>
<td><strong>CRTY</strong></td>
<td>CURSOR_TYPE</td>
<td>CURSOR</td>
<td>Select the type of cursor.</td>
</tr>
<tr>
<td><strong>CRVA?</strong></td>
<td>CURSOR_VALUE?</td>
<td>CURSOR</td>
<td>Returns trace values measured by specified cursors.</td>
</tr>
<tr>
<td><strong>CSV</strong></td>
<td>CSV_SAVE</td>
<td>SAVE/RECALL</td>
<td>Saves specified waveform data of CSV format to USB device.</td>
</tr>
<tr>
<td><strong>CYMT</strong></td>
<td>CYMOMETER</td>
<td>MEASURE</td>
<td>Returns the current cymometer value which displaying on the screen.</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>DATE</td>
<td>SYSTEM</td>
<td>Changes the date/time of the internal real-time clock.</td>
</tr>
<tr>
<td><strong>DDR?</strong></td>
<td>DDR?</td>
<td>STATUS</td>
<td>Clears the Device Dependent Register (DDR).</td>
</tr>
<tr>
<td><strong>DEF</strong></td>
<td>DEFINE?</td>
<td>MATH</td>
<td>Specifies math expression for function evaluation.</td>
</tr>
<tr>
<td><strong>DTIJN</strong></td>
<td>DOT_JOIN</td>
<td>DISPLAY</td>
<td>Controls the interpolation lines between data points.</td>
</tr>
<tr>
<td>*<strong>ESE</strong></td>
<td>*ESE</td>
<td>STATUS</td>
<td>Sets the Standard Event Status Enable register (ESE).</td>
</tr>
<tr>
<td>*<strong>ESR?</strong></td>
<td>*ESR?</td>
<td>STATUS</td>
<td>Reads, clears the Event Status Register (ESR).</td>
</tr>
<tr>
<td><strong>EXR?</strong></td>
<td>EXR?</td>
<td>STATUS</td>
<td>Reads, clears the Execution error Register (EXR).</td>
</tr>
<tr>
<td><strong>FPAR</strong></td>
<td>FRAME_PARAM</td>
<td>HISTORY</td>
<td>Get frame param.</td>
</tr>
<tr>
<td><strong>FRAM</strong></td>
<td>FRAME_SET</td>
<td>HISTORY</td>
<td>History Frame No. set.</td>
</tr>
<tr>
<td><strong>FTIM</strong></td>
<td>FRAME_TIME</td>
<td>HISTORY</td>
<td>Get frame Acq. Time.</td>
</tr>
<tr>
<td><strong>FILT</strong></td>
<td>FILTER</td>
<td>FUNCTION</td>
<td>Enables or disables the filter of specified source.</td>
</tr>
<tr>
<td><strong>FILTS</strong></td>
<td>FILT_SET</td>
<td>FUNCTION</td>
<td>Selects the type of filter, and sets the limit value of filter.</td>
</tr>
<tr>
<td><strong>FFT</strong></td>
<td>FFT_WINDOW</td>
<td>MATH</td>
<td>Selects the window of FFT.</td>
</tr>
<tr>
<td><strong>FFTZ</strong></td>
<td>FFT_ZOOM</td>
<td>MATH</td>
<td>Selects the zoom in/out</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Category</td>
<td>Notes</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>FFTS</td>
<td>FFT_SCALE</td>
<td>MATH</td>
<td>Selects the vertical scale of FFT trace.</td>
</tr>
<tr>
<td>FFTU</td>
<td>FFT_UNIT</td>
<td>MATH</td>
<td>Selects the vertical scale unit of FFT trace.</td>
</tr>
<tr>
<td>FFTT</td>
<td>FFT_TDIV</td>
<td>MATH</td>
<td>Selects the horizontal scale of FFT trace.</td>
</tr>
<tr>
<td>FTPP</td>
<td>FFT_POSITION</td>
<td>MATH</td>
<td>Selects the position of FFT trace.</td>
</tr>
<tr>
<td>FFTE</td>
<td>FFT_CENTER</td>
<td>MATH</td>
<td>Selects the center frequency of FFT trace.</td>
</tr>
<tr>
<td>FFTF</td>
<td>FFT_FULLSCREEN</td>
<td>MATH</td>
<td>Enables or disables to display the FFT trace full screen.</td>
</tr>
<tr>
<td>GRDS</td>
<td>GRID_DISPLAY</td>
<td>DISPLAY</td>
<td>Selects the type of grid.</td>
</tr>
<tr>
<td>HMAG</td>
<td>HOR_MAGNIFY</td>
<td>ZOOM</td>
<td>Horizontally expands the selected expansion trace.</td>
</tr>
<tr>
<td>HPOS</td>
<td>HOR_POSITION</td>
<td>ZOOM</td>
<td>Horizontally positions intensified zone’s center.</td>
</tr>
<tr>
<td>INTS</td>
<td>INTENSITY</td>
<td>DISPLAY</td>
<td>Sets the grid or trace/text intensity level.</td>
</tr>
<tr>
<td>INR?</td>
<td>INR?</td>
<td>STATUS</td>
<td>Reads, clears INternal state change Register (INR).</td>
</tr>
<tr>
<td>INVS</td>
<td>INVERT_SET</td>
<td>DISPLAY</td>
<td>Invert the trace or the math waveform of specified source.</td>
</tr>
<tr>
<td>MTVP</td>
<td>MATH_VERT_POS</td>
<td>MATH</td>
<td>Controls the vertical position of math waveform of specified source.</td>
</tr>
<tr>
<td>MTVD</td>
<td>MATH_VERT_DIV</td>
<td>MATH</td>
<td>Controls the vertical sensitivity of math waveform of specified source.</td>
</tr>
<tr>
<td>MEAD</td>
<td>MEASURE_DELY</td>
<td>MEASURE</td>
<td>Controls the type of delay measure.</td>
</tr>
<tr>
<td>MENU</td>
<td>MENU</td>
<td>DISPLAY</td>
<td>Enables or disables to display the current menu.</td>
</tr>
<tr>
<td>MSIZ</td>
<td>MEMORY_SIZE</td>
<td>ACQUISITION</td>
<td>Returns the maximal memory size.</td>
</tr>
<tr>
<td>OFST</td>
<td>OFFSET</td>
<td>CHANNEL</td>
<td>Allows output channel vertical offset adjustment.</td>
</tr>
</tbody>
</table>
## Digital Oscilloscopes Series

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*OPC</td>
<td>*OPC</td>
<td>STATUS</td>
<td>Sets the OPC bit in the Event Status Register (ESR).</td>
</tr>
<tr>
<td>PACL</td>
<td>PARAMETER_CLR</td>
<td>PASS/FAIL</td>
<td>Clears all current parameters in Custom, Pass/Fail.</td>
</tr>
<tr>
<td>PACU</td>
<td>PARAMETER_CUSTOM</td>
<td>MEASURE</td>
<td>Controls parameters with customizable qualifiers.</td>
</tr>
<tr>
<td>PAVA?</td>
<td>PARAMETER_VALUE?</td>
<td>MEASURE</td>
<td>Returns current parameter, mask test values.</td>
</tr>
<tr>
<td>PESU</td>
<td>PERSIST_SETUP</td>
<td>DISPLAY</td>
<td>Selects display persistence duration.</td>
</tr>
<tr>
<td>PNSU</td>
<td>PANEL_SETUP</td>
<td>SAVE/RECALL</td>
<td>Complements the *SAV/*RST commands.</td>
</tr>
<tr>
<td>PFDS</td>
<td>PF_DISPLAY</td>
<td>PASS/FAIL</td>
<td>Enables or disables to display the test and the message options of pass/fail.</td>
</tr>
<tr>
<td>PFST</td>
<td>PF_SET</td>
<td>PASS/FAIL</td>
<td>Sets the X mask and the Y mask.</td>
</tr>
<tr>
<td>PFSL</td>
<td>PF_SAVLOAD</td>
<td>PASS/FAIL</td>
<td>Saves or recalls the created mask setting.</td>
</tr>
<tr>
<td>PFCT</td>
<td>PF_CONTROL</td>
<td>PASS/FAIL</td>
<td>Selects the “operate”, “output” and the “stop on output” which are the options of pass/fail.</td>
</tr>
<tr>
<td>PFSC</td>
<td>PF_SOURCE</td>
<td>PASS/FAIL</td>
<td>Selects the source of pass/fail.</td>
</tr>
<tr>
<td>PFBF</td>
<td>PF_BUFFER</td>
<td>PASS/FAIL</td>
<td>Selects the “output” which is the options of pass/fail.</td>
</tr>
<tr>
<td>PFSS</td>
<td>PF_FAIL_STOP</td>
<td>PASS/FAIL</td>
<td>Selects the “stop on fail” which is the options of pass/fail.</td>
</tr>
<tr>
<td>PFOP</td>
<td>PF_OPERATION</td>
<td>PASS/FAIL</td>
<td>Selects the “operate” which is the options of pass/fail.</td>
</tr>
<tr>
<td>PFCM</td>
<td>PF_CREATEM</td>
<td>PASS/FAIL</td>
<td>Creates the mask of the pass/fail.</td>
</tr>
<tr>
<td>PFDD</td>
<td>PFDATEDIS</td>
<td>PASS/FAIL</td>
<td>Return the number of the pass/fail monitor which can be displayed on the screen.</td>
</tr>
<tr>
<td>*RCL</td>
<td>*RCL</td>
<td>SAVE/RECALL</td>
<td>Recalls one of five non-volatile panel setups.</td>
</tr>
<tr>
<td>REC</td>
<td>RECALL</td>
<td>SAVE/RECALL</td>
<td>Recalls a waveform file from the current directory.</td>
</tr>
</tbody>
</table>
### Digital Oscilloscopes Series

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCPN</td>
<td>RECALL_PANEL</td>
<td>SAVE/RECALL</td>
</tr>
<tr>
<td>*RST</td>
<td>*RST</td>
<td>SAVE/RECALL</td>
</tr>
<tr>
<td>REFCL</td>
<td>REF_CLOSE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFDI</td>
<td>REF_DISPALY</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFLA</td>
<td>REF_LOCATION</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFPO</td>
<td>REF_POISSION</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFSA</td>
<td>REF_SAVE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFSC</td>
<td>REF_SCLALE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFSR</td>
<td>REF_SOURCE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>REFS</td>
<td>REF_SET</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>*SAV</td>
<td>*SAV</td>
<td>SAVE/RECALL</td>
</tr>
<tr>
<td>SCDP</td>
<td>SCREEN_DUMP</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>SCSV</td>
<td>SCREEN_SAVE</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>*SRE</td>
<td>*SRE</td>
<td>STATUS</td>
</tr>
<tr>
<td>STOP</td>
<td>STOP</td>
<td>TRIGGER</td>
</tr>
<tr>
<td>STPN</td>
<td>STORE_PANEL</td>
<td>SAVE/RECALL</td>
</tr>
<tr>
<td>SAST</td>
<td>SAMPLE_STATUS</td>
<td>ACQUISITION</td>
</tr>
<tr>
<td>SARA</td>
<td>SAMPLE_RATE</td>
<td>ACQUISITION</td>
</tr>
<tr>
<td>SANU</td>
<td>SAMPLE_NUM</td>
<td>ACQUISITION</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SET50</td>
<td>Sets the trigger level of the trigger source to the centre of the signal amplitude.</td>
<td></td>
</tr>
<tr>
<td>SKEW</td>
<td>Sets the skew of specified trace.</td>
<td></td>
</tr>
<tr>
<td>SXSA</td>
<td>Sets the type of the interpolation.</td>
<td></td>
</tr>
<tr>
<td>TDIV</td>
<td>Modifies the time base setting.</td>
<td></td>
</tr>
<tr>
<td>TMPL</td>
<td>Produces a complete waveform template copy.</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>Enables or disables the display of a trace.</td>
<td></td>
</tr>
<tr>
<td>TRCP</td>
<td>Sets the coupling mode of the specified trigger source.</td>
<td></td>
</tr>
<tr>
<td>TRDL</td>
<td>Sets the time at which the trigger is to occur.</td>
<td></td>
</tr>
<tr>
<td>TRLV</td>
<td>Adjusts the trigger level of the specified trigger source.</td>
<td></td>
</tr>
<tr>
<td>TRLV2</td>
<td>Adjusts the second trigger level of the specified trigger source.</td>
<td></td>
</tr>
<tr>
<td>TRMD</td>
<td>The trigger mode.</td>
<td></td>
</tr>
<tr>
<td>TRSE</td>
<td>Selects the condition that will trigger acquisition.</td>
<td></td>
</tr>
<tr>
<td>TRSL</td>
<td>Sets the trigger slope of the specified trigger source.</td>
<td></td>
</tr>
<tr>
<td>TRWI</td>
<td>Return relative height of the trigger window</td>
<td></td>
</tr>
<tr>
<td>TRPA</td>
<td>Sets the condition of the pattern trigger</td>
<td></td>
</tr>
<tr>
<td>UNIT</td>
<td>Sets the unit of specified trace.</td>
<td></td>
</tr>
<tr>
<td>VPOS</td>
<td>Adjusts the vertical position of the FFT trace.</td>
<td></td>
</tr>
<tr>
<td>VDIV</td>
<td>Sets the vertical sensitivity.</td>
<td></td>
</tr>
<tr>
<td>WF</td>
<td>Gets the waveform from the instrument.</td>
<td></td>
</tr>
<tr>
<td>WFSU</td>
<td>Specifies amount of waveform data to go to controller.</td>
<td></td>
</tr>
<tr>
<td>SETTO%50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKEW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINXX_SAMPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME_DIV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMPLATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_COUPLING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_DELAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_LEVEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_LEVEL2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_SELECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_SLOPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_WINDOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIG_PATTERN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERT_POSITION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOLT_DIV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAVEFORM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAVEFORM_SETUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAVEFORM_TRANS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Digital Oscilloscopes Series
<table>
<thead>
<tr>
<th>XYDS</th>
<th>XY_DISPLAY</th>
<th>DISPLAY</th>
<th>Enables or disables to display the XY format</th>
</tr>
</thead>
</table>

Digital Oscilloscopes Series
### Commands & Queries

#### ACQUISITION

**ACQUIRE_WAY, ACQW**

**Command / Query**

**DESCRIPTION**

The ACQUIRE_WAY command specifies the acquisition mode.

The ACQUIRE_WAY? Query returns the current acquisition mode.

**COMMAND SYNTAX**

```
ACQUIRE_WAY <mode>[,<time>]
```

- `<mode>` := \{SAMPLING, PEAK_DETECT, AVERAGE, HIGH_RES \}
- `<time>` := \{4, 16, 32, 64, 128, 256, 512, etc\}

**Note:**

1. The [HIGH_RES] option of mode is applicable for SPO models.
2. The `<time>` parameter only can be set with the average acquisition mode. And its options vary with model.

**QUERY SYNTAX**

```
ACQUIRE_WAY?
```

**RESPONSE FORMAT**

```
ACQUIRE_WAY <mode>[,<time>]
```

**EXAMPLE**

The following command sets the acquisition mode to average mode and also sets the average value to 16.

Command message:

ACQW AVERAGE, 16

**RELATED COMMANDS**

AVGA,
The ALL_STATUS? Query reads and clears the contents of all status registers: STB, ESR, INR, DDR, CMR, EXR and URR except for the MAV bit (bit 6) of the STB register. For an interpretation of the contents of each register, refer to the appropriate status register.

The ALL_STATUS? Query is useful in a complete overview of the state of the instrument.

**QUERY SYNTAX**

```
ALL_STATUS?
```

**RESPONSE FORMAT**

```
ALL_STATUS
STB,<value>,ESR,<value>,INR,<value>,DDR,<value>,CMR,<value>,EXR,<value>,URR,<value>
```

\(<value> : = 0 \text{ to } 65535\)

**EXAMPLE**

The following instruction reads the contents of all the status registers:

Command message:
```
ALST?
```

Response message:
```
ALST STB, 0, ESR, 52, INR, 5, DDR, 0, CMR, 4, EXR, 24, URR, 0
```

**RELATED COMMANDS**

*CLS, CMR?, DDR?, *ESR?, EXR?, *STB?, URR?
The ARM ACQUISITION command enables the signal acquisition process by changing the acquisition state (trigger mode) from “stopped” to “single”.

**COMMAND SYNTAX**

ARM acquisition

**EXAMPLE**

The following command enables signal acquisition:

Command message:

ARM

**RELATED COMMANDS**

STOP, *TRG, TRIG_MODE, WAIT
**CHANNEL**

**ATTENUATION, ATTN**
Command /Query

**DESCRIPTION**
The ATTENUATION command selects the vertical attenuation factor of the probe. Values of 1, 5, 10, 50, 100, 500, and 1000 may be specified.

The ATTENUATION? Query returns the attenuation factor of the specified channel.

**COMMAND SYNTAX**

\[
\text{<channel>: ATTeNuation} \ <\text{attenuation>}
\]

\[
\text{<channel>} := \{C1, C2, C3, C4\}
\]

\[
\text{<attenuation>} := \{0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}
\]

**QUERY SYNTAX**

\[
\text{<channel>: ATTeNuation?}
\]

**RESPONSE FORMAT**

\[
\text{<channel>: ATTeNuation} \ <\text{attenuation>}
\]

**EXAMPLE**
The following command sets to 100 the attenuation factor of Channel 1:

Command message:

C1:ATTN 100
**SYSTEM**

**DESCRIPTION**

The AUTO_CALIBRATE command is used to enable or disable the quick calibration of the instrument.

The quick calibration may be disabled by issuing the command ACAL OFF. Whenever it is convenient, a *CAL? Query may be issued to fully calibrate the oscilloscope.

The response to the AUTO_CALIBRATE? Query indicates whether quick-calibration is enabled.

This command is only used in the CFL series of instruments.

**COMMAND SYNTAX**

Auto_CALibrate <state>

<state> := {ON, OFF}

**QUERY SYNTAX**

Auto_CALibrate?

**RESPONSE FORMAT**

Auto_CALibrate <state>

**EXAMPLE**

The following instruction disables quick calibration:

Command message: ACAL OFF

**RELATED COMMANDS**

*CAL?
**ACQUISITION**

**AUTO_SETUP, ASET**

Command

**DESCRIPTION**

The AUTO_SETUP command attempts to identify the waveform type and automatically adjusts controls to produce a usable display of the input signal.

**COMMAND SYNTAX**

AUTO_SETUP

**EXAMPLE**

The following command instructs the oscilloscope to perform an auto-setup:

Command message: ASET

**RELATED COMMANDS**

AUTTS
**ACQUISITION**

**AUTO_TYPESET, AUTTS**  
Command /Query

**DESCRIPTION**  
The AUTO_TYPESET command selects the specified type of automatically adjusting which is used to display.

**COMMAND SYNTAX**  
AUTO_TYPESET <type>  
<type> := {SP,MP,RS,DRP,RC}  
SP means only one period to be displayed, MP means multiple periods to be displayed, RS means the waveform is triggered on the rise side, DRP means the waveform is triggered on the drop side, and RC means to go back to the state before auto set.

**QUERY SYNTAX**  
AUTO_TYPESET?

**RESPONSE FORMAT**  
AUTO_TYPESET <type>

**EXAMPLE**  
The following command sets the type of automatic adjustment to multiple periods:

Command message:  
AUTTS MP

**RELATED COMMANDS**  
ASET
**ACQUISITION**

**AVERAGE_ACQUIRE, AVGA**

**Command / Query**

**DESCRIPTION**
The AVERAGE_ACQUIRE command selects the average times of average acquisition.

The response to the AVERAGE_ACQUIRE query indicates the times of average acquisition.

**COMMAND SYNTAX**

AVERAGE_ACQUIRE <time>

<time> := {4, 16, 32, 64, 128, 256, etc}

**Note:**
The <time> parameter’s options vary with model.

**QUERY SYNTAX**

AVERAGE_ACQUIRE?

**RESPONSE FORMAT**

AVERAGE_ACQUIRE <time>

**EXAMPLE**
The following turns the average times of average acquisition 16:

Command message:
AVGA 16
ACQUISITION

BANDWIDTH_LIMIT, BWL
Command/Query

DESCRIPTION

BANDWIDTH_LIMIT enables or disables the bandwidth-limiting low-pass filter. If the bandwidth filters are on, it will limit the bandwidth to reduce display noise. When you turn Bandwidth Limit ON, the Bandwidth Limit value is set to 20 MHz. It also filters the signal to reduce noise and other unwanted high frequency components.

The response to the BANDWIDTH_LIMIT? Query indicates whether the bandwidth filters are on or off.

COMMAND SYNTAX

BandWidth_Limit <channel>, <mode>
[<channel>, <mode> [, <channel>, <mode> [, <channel>, <mode>]]]

<channel> = {C1, C2, C3, C4}
<mode> = {ON, OFF}

QUERY SYNTAX

BandWidth_Limit?

RESPONSE FORMAT

BandWidth_Limit <channel>, <mode>
[<channel>, <mode> [, <channel>, <mode> [, <channel>, <mode>]]]

EXAMPLE

The following turns on the bandwidth filter for all channels, when Global_BWL is on (as it is by default)

The following turns the bandwidth filter on for Channel 1 only:

Command message:
BWL C1, ON
**BUZZER, BUZZ**  
Command /Query

| DESCRIPTION | The BUZZER command enables or disables sounds for keypresses and other functions.
|             | The response to the BUZZER? query indicates whether the sound switch is enabled or not.
| COMMAND SYNTAX | BUZZer <state>
| <state> | : {ON, OFF}
| QUERY SYNTAX | BUZZER?
| RESPONSE FORMAT | BUZZER <state>
| EXAMPLE | Sending the following code will enable the oscilloscope sound.
|          | Command message:
|          | BUZZ ON |
MISCELLANEOUS

*CAL?
Query

DESCRIPTION

The *CAL? query causes the oscilloscope to perform an internal self-calibration and generates a response.

QUERY SYNTAX

*CAL?

RESPONSE FORMAT

*CAL <diagnostics>
<diagnostics> := 0
0 = Calibration successful

EXAMPLE

The following instruction forces a self-calibration:

Command message:
*CAL?

Response message:
*CAL 0

RELATED COMMANDS

AUTO_CALIBRATE
COMMUNICATION

DESCRIPTION
The COMM_HEADER command controls the way the oscilloscope formats responses to queries. There are three response formats: LONG, in which responses start with the long form of the header word; SHORT, where responses start with the short form of the header word; and OFF, for which headers are omitted from the response and units in numbers are suppressed.

Unless you request otherwise, the SHORT response format is used.

This command does not affect the interpretation of messages sent to the oscilloscope. Headers can be sent in their long or short form regardless of the COMM_HEADER setting.

Querying the vertical sensitivity of Channel 1 may result in one of the following responses:

<table>
<thead>
<tr>
<th>COMM_HEADER</th>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONG</td>
<td>C1:VOLT_DIV 200E-3 V</td>
</tr>
<tr>
<td>SHORT</td>
<td>C1:VDIV 200E-3 V</td>
</tr>
<tr>
<td>OFF</td>
<td>200E-3</td>
</tr>
</tbody>
</table>

COMMAND SYNTAX
Comm_HeaDeR <mode>
<mode> : = {SHORT, LONG, OFF}

QUERY SYNTAX
Comm_HeaDeR?

RESPONSE FORMAT
Comm_HeaDeR <mode>

EXAMPLE
The following code sets the response header format to SHORT:

Command message:
CHDR SHORT
### STATUS

<table>
<thead>
<tr>
<th>Command</th>
<th>DESCRIPTION</th>
<th>COMMAND SYNTAX</th>
<th>EXAMPLE</th>
<th>RELATED COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CLS</td>
<td>The *CLS command clears all the status data registers.</td>
<td>*CLS</td>
<td>The following command causes all the status data registers to be cleared:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*CLS</td>
<td>Command message:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*CLS</td>
<td>*CLS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALR, STATUS, CMR, DDR, *ESR, EXR, *STB, URR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Unrecognized command/query header</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Invalid character</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Invalid separator</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Missing parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Unrecognized keyword</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>String error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Parameter cannot allowed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Command String Too Long</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Query cannot allowed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Missing Query mask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Invalid parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Parameter syntax error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Filename too long</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**DESCRIPTION**

The COMM_NET command changes the IP address of the oscilloscope’s internal network interface.

The COMM_NET? query returns the IP address of the oscilloscope’s internal network interface.

**COMMAND SYNTAX**

```
COMM_NET <ip_add0>, <ip_add1>, <ip_add2>, <ip_add3>
< ip_add >:= 0 to 255
```

**QUERY SYNTAX**

```
COMM_NET?
```

**RESPONSE FORMAT**

```
COMM_NET <ip_add0>, <ip_add1>, <ip_add2>, <ip_add3>
```

**EXAMPLE**

This instruction will change the IP address to 10.11.0.230:

Command message:
CONET 10,11,0,230
### Function

#### Description
The COUNTER command enables or disables the cymometer display on the screen of instrument.

The response to the COUNTER? query indicates whether the cymometer is displayed on the screen of instrument.

<table>
<thead>
<tr>
<th>Command Syntax</th>
<th>COUNTER &lt;state&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; state&gt;</td>
<td>{ON, OFF}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Syntax</th>
<th>COUNTER?</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Response Format</th>
<th>COUNTER &lt;state&gt;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>The following command enables the cymometer display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command message</td>
<td>COUN ON</td>
</tr>
</tbody>
</table>

**Note:**
This command is suitable for non-SPO models.
**ACQUISITION**

**COUPLING, CPL**

**Command/Query**

**DESCRIPTION**

The COUPLING command selects the coupling mode of the specified input channel.

The COUPLING? query returns the coupling mode of the specified channel.

**COMMAND SYNTAX**

<channel>: CouPLing <coupling>

<channel>: = {C1, C2, C3, C4}

<coupling>: = {A1M, A50, D1M, D50, GND}

The A of the <coupling> is alternating current. The D of the <coupling> is direct current. 1M and 50 is the impedance of input. Some series (CML) couldn’t have the set of input impedance.

**Note:**

The options of <coupling> vary with models. If your oscilloscope is an SPO model, the options are {A1M, A50, D1M, D50, GND}, otherwise the options are {A1M, D1M, GND}.

**QUERY SYNTAX**

<channel>: CouPLing?

**RESPONSE FORMAT**

<channel>: CouPLing <coupling>

**EXAMPLE**

The following command sets the coupling of Channel 2 to 50 ΩDC:

Command message:
C2:CPL D50
CURSOR

CURSOR_AUTO,CRAU
Command

DESCRIPTION
The CURSOR_AUTO command changes the cursor mode to auto mode.

COMMAND SYNTAX
CRAU

EXAMPLE
The following code changes the cursor mode to auto mode:

Command message:
CRAU

Note:
This command is suitable for non-SPO models.
**CURSOR**

**CURSOR_MEASURE, CRMS**

Command / Query

**DESCRIPTION**

The CURSOR_MEASURE command specifies the mode of cursor or parameter measurement to be displayed.

The CURSOR_MEASURE? query indicates which cursors or parameter measurements are currently displayed.

**COMMAND SYNTAX**

CuRsr_MeaSure <mode>

Format 1:

<mode>={OFF, ON}

Format 2:

<mode>={OFF, MANUAL, TRACK}

**Note:**

1. If the oscilloscope doesn’t have auto cursor, you should use format 1. OFF means manual mode, ON means track mode.
2. If the oscilloscope has auto cursor, you should use format 2.

**QUERY SYNTAX**

CuRsr_MeaSure?

**RESPONSE FORMAT**

CuRsr_MeaSure <mode>

**EXAMPLE**

The following command determines cursor function is turned off:

Command message:

CRMS OFF

**RELATED COMMANDS**

CURSOR_VALUE, PARAMETER_VALUE
**CURSOR**

**DESCRIPTION**

The CURSOR_SET command allows the user to position any one of the eight independent cursors at a given screen location. The positions of the cursors can be modified or queried even if the required cursor is not currently displayed on the screen. When setting a cursor position, a trace must be specified, relative to which the cursor will be positioned.

The CURSOR_SET? Query indicates the current position of the cursor(s). The values returned depend on the grid type selected.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VREF</td>
<td>The voltage-value of curA under manual cursor mode.</td>
</tr>
<tr>
<td>VDIF</td>
<td>The voltage-value of curB under manual cursor mode.</td>
</tr>
<tr>
<td>TREF</td>
<td>The time value of curA under manual cursor mode.</td>
</tr>
<tr>
<td>TDIF</td>
<td>The time value of curB under manual cursor mode.</td>
</tr>
<tr>
<td>HREF</td>
<td>The time value of curA under Track cursor mode.</td>
</tr>
<tr>
<td>HDIF</td>
<td>The time value of curB under Track cursor mode.</td>
</tr>
</tbody>
</table>

**COMMAND SYNTAX**

```plaintext
<trace>:CuRsor_SeT<cursor>,<position>[,<cursor>,<position>,<cursor>,<position>]
```

- `<trace>`: = {C1, C2, C3, C4}
- `<cursor>`: = { VREF, VDIF, TREF, TDIF, HREF, HDIF }
- `<position>`(horizontal): = {-7 to 7 DIV }
- `<position>`(vertical): = {-4 to 4 DIV }

**Note:**
1. The horizontal position’s value is related to the size of screen. For SPO models, the position’s value is in the range of -7 to 7 and you need add the unit (DIV) to the value. In non-SPO models
2. The vertical position’s value is related to the size of screen. For SPO models, you need add the unit (DIV) to the value.

**QUERY SYNTAX**

```
<trace>: CuRsr_SeT? [<cursor>, …<cursor>]  
<cursor>:=  
{ VREF,VDIF,TREF,TDIF,HREF,HDIF}
```

**RESPONSE FORMAT**

```
<trace>: CuRsr_SeT <cursor>, <position> [,  
<cursor>, <position>, <cursor>, <position>]
```

**EXAMPLE**

The following command positions the VREF and VDIF cursors at +3 DIV and −1 DIV respectively, using C1 as a reference:

Command message:

```
C1: CRST VREF, 3DIV, VDIF, −1DIV
```

**RELATED COMMANDS**

CURSOR_MEASURE, CURSOR_VALUE, PARAMETER_VALUE
**CURSOR**

**DESCRIPTION**

The CURSOR_VALUE? query returns the values measured by the specified cursors for a given trace. (The PARAMETER_VALUE? query is used to obtain measured waveform parameter values.)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HREL</td>
<td>the delta time-value, reciprocal of delta time-value, curA time-value and curB time-value.</td>
</tr>
<tr>
<td>VREL</td>
<td>the delta volt-value, curA volt-value and curB volt-value under manual cursor mode</td>
</tr>
</tbody>
</table>

**Note:**

For non-SPO models, VREL is the delta volt-value under manual cursor mode.

**QUERY SYNTAX**

<trace>: CuRsor_Value? [<mode>,...<mode>]
<trace>: = { C1, C2, C3, C4}
<mode>: = { HREL, VREL }

**RESPONSE FORMAT**

<trace>: CuRsor_Value HREL,
<delta_hori>,< 1/delta_hori >,< curA _value >,
<curB _value >
<trace>: CuRsor_Value VREL,<delta_vert>,< curA _value >,< curB _value >

**EXAMPLE**

The following query reads the vertical value under manual cursor mode (VREL) on Channel 2:

Command message:
C2:CRVA? VREL

Response message:
C2:CuRsor_Value VREL,3.96v,-0.02v,-3.98v

**RELATED COMMANDS**

CURSOR_SET, PARAMETER_VALUE
```markdown
**CURSOR**

**CURSOR_TYPE, CRTY**

Command / Query

**DESCRIPTION**

The CURSOR_TYPE command specifies the type of cursor to be displayed.

**COMMAND SYNTAX**

CURSOR_TYPE <type>

<mode>={X, Y,X-Y}

**QUERY SYNTAX**

CURSOR_TYPE?

**RESPONSE FORMAT**

CURSOR_TYPE <type>

**EXAMPLE**

The following command determines cursor type is Y:

Command message:
CRTY Y

**RELATED COMMANDS**

CURSOR_MEASURE

Note:
This command is suitable for SPO models.
**SAVE/RECALL**

**CSV_SAVE, CSVS**

**Command /Query**

**DESCRIPTION**

The CSV_SAVE command selects the specified option of storing CSV format waveform.

The CSV_SAVE? query returns the option of storing waveform data of CSV format.

**COMMAND SYNTAX**

Format1:
CSV_SAVE SAVE,<state>

The option SAVE is that if the waveform data is stored with parameter.

<save> := \{OFF, ON\}

Format2:
CSV_SAVE DD,<DD>,SAVE,<state>

The option DD is the data depth which is saved as. The option SAVE is that if the waveform data is stored with parameter.

<DD> := \{MAX, DIS\} the meaning of MAX is saved as the maximum data depth. The meaning of DIS is saved as the date depth which is displayed on the screen

<save> := \{OFF, ON\}

**Note:**

This command varies with models, so there are two formats. If your oscilloscope can set the data depth of CSV file which will be saved, you should use Format2, such as non-SPO models, otherwise you should use Format1.

**QUERY SYNTAX**

CSV_SAVE?

**RESPONSE FORMAT**

CSV_SAVE SAVE, <state>

**EXAMPLE**

The following command sets “para” save to off

Command message:

Format1:
CSV_SAVE SAVE,OFF

Format2:
CSVS DD,DIS,SAVE,OFF
FUNCTION CYMOMETER?, CYMT?

DESCRIPTION
The response to the CYMOMETER? query is the value of cymometer which displaying on the screen of the instrument. When the signal frequency is less than 10Hz, it returns 10Hz.

QUERY SYNTAX
CYMOMETER?

RESPONSE FORMAT
CYMOMETER <option>

EXAMPLE
The following instruction returns the value of cymometer which displaying on the screen of the instrument.

Response message:
CYMT 10Hz
**SYSTEM**

**DATE**

Command /Query

**DESCRIPTION**

The DATE command changes the date/time of the oscilloscope’s internal real-time clock.

The command is only used in the CFL series instrument.

**COMMAND SYNTAX**

DATE <day>, <month>, <year>, <hour>, <minute>, <second>

- `<day>`: = 1 to 31
- `<month>`: = {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC}
- `<year>`: = 1990 to 2089
- `<hour>`: = 0 to 23
- `<minute>`: = 0 to 59
- `<second>`: = 0 to 59

**QUERY SYNTAX**

DATE?

**RESPONSE FORMAT**

DATE <day>, <month>, <year>, <hour>, <minute>, <second>

**EXAMPLE**

This instruction will change the date to NOV. 1, 2009 and the time to 14:38:16:

Command message:
DATE 1, NOV, 2009,14,38,16

**Note:**

This command is suitable for the model which has this function.
**STATUS**

**DESCRIPTION**
The DDR? Query reads and clears the contents of the Device Dependent or device specific error Register (DDR). In the case of a hardware failure, the DDR register specifies the origin of the failure.

**QUERY SYNTAX**

**RESPONSE FORMAT**

DDR <value>

<value>: = 0 to 65535

**EXAMPLE**
The following instruction reads the contents of the DDR register:

Command message:

DDR?

Response message:

DDR 0

**RELATED COMMANDS**

ALL_STATUS? ,*CLS
**FUNCTION**

**DEFINE, DEF**
Command / Query

**DESCRIPTION**

The DEFINE command specifies the mathematical expression to be evaluated by a function.

**COMMAND SYNTAX**

DEFine EQN,’<equation>’

*Note:*

<equation> is the mathematical expression

<table>
<thead>
<tr>
<th>Function Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source1&gt; + &lt;source2&gt;</td>
</tr>
<tr>
<td>Addition</td>
</tr>
<tr>
<td>&lt;source1&gt; - &lt;source2&gt;</td>
</tr>
<tr>
<td>Subtraction</td>
</tr>
<tr>
<td>&lt;source1&gt;*&lt;source2&gt;</td>
</tr>
<tr>
<td>Multiplication</td>
</tr>
<tr>
<td>&lt;source1&gt;/&lt;source2&gt;</td>
</tr>
<tr>
<td>Ratio</td>
</tr>
<tr>
<td>FFT(source x)</td>
</tr>
<tr>
<td>FFT</td>
</tr>
<tr>
<td>INTG(source x)</td>
</tr>
<tr>
<td>Integral</td>
</tr>
<tr>
<td>DIFF(source x)</td>
</tr>
<tr>
<td>Differentiator</td>
</tr>
<tr>
<td>SQRT(source x)</td>
</tr>
<tr>
<td>Square Root</td>
</tr>
</tbody>
</table>

**QUERY SYNTAX**

DEFine?

**RESPONSE FORMAT**

DEFine EQN,’<equation>’

**EXAMPLE**

Command message:

DEFine EQN,’C1*C2’
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>The DOJOIN command controls the interpolation lines between data points.</th>
</tr>
</thead>
</table>
| COMMAND SYNTAX | DoT_JoiN <state>  
<state> = {ON, OFF} |
| QUERY SYNTAX | DoT_JoiN? |
| RESPONSE FORMAT | DoT_JoiN <state> |
| EXAMPLE | The following instruction turns off the interpolation lines:  
Command message:  
DTJN OFF |
### STATUS

#### *ESE Command /Query

**DESCRIPTION**
The *ESE command sets the Standard Event Status Enable register (ESE). This command allows one or more events in the ESR register to be reflected in the ESB summary message bit (bit 5) of the STB register.

**COMMAND SYNTAX**
*ESE <value>

*value*: = 0 to 255

**QUERY SYNTAX**
*ESE?

**RESPONSE FORMAT**
*ESE <value>

**EXAMPLE**
The following instruction allows the ESB bit to be set if a user request (URQ bit 6, i.e. decimal 64) and/or a device dependent error (DDE bit 3, i.e. decimal 8) occurs. Summing these values yields the ESE register mask 64+8=72.

Command message:
*ESE 72

**RELATED COMMANDS**
*ESR
**DESCRIPTION**

The *ESR? query reads and clears the contents of the Event Status Register (ESR). The response represents the sum of the binary values of the register bits 0 to 7.

**QUERY SYNTAX**

*ESR?

**RESPONSE FORMAT**

*ESR<value>

<value>: = 0 to 255

**EXAMPLE**

The following instruction reads and clears the contents of the ESR register:

Command message:

*ESR?

Response message:

*ESR 0

**RELATED COMMANDS**

ALL_STATUS, *CLS, *ESE

**ADDITIONAL INFORMATION**

### Standard Event Status Register (ESR)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Value</th>
<th>Bit Name</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>15~8</td>
<td>reserved by IEEE 488.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>PON</td>
<td>Power off-to-ON transition as occurred (1)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>URQ</td>
<td>User Request has been issued (2)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>CME</td>
<td>Command parser Error has been detected (3)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>EXE</td>
<td>Execution Error detected (4)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>DDE</td>
<td>Device specific Error occurred (5)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>QYE</td>
<td>Query Error occurred (6)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>RQC</td>
<td>Instrument never requests bus control (7)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OPC</td>
<td>Instrument never requests bus control (8)</td>
<td></td>
</tr>
</tbody>
</table>
Notes

(1) The Power On (PON) bit is always turned on (1) when the unit is powered up.

(2) The User Request (URQ) bit is set true (1) when a soft key is pressed. An associated register URR identifies which key was selected. For further details refer to the URR? query.

(3) The Command parser Error bit (CME) is set true (1) whenever a command syntax error is detected. The CME bit has an associated Command parser Register (CMR) which specifies the error code. Refer to the query CMR? for further details.

(4) The Execution Error bit (EXE) is set true (1) when a command cannot be executed due to some device condition (e.g. oscilloscope in local state) or a semantic error. The EXE bit has an associated Execution Error Register (EXR) which specifies the error code. Refer to query EXR? for further details.

(5) The Device specific Error (DDE) is set true (1) whenever a hardware failure has occurred at power-up, or execution time, such as a channel overload condition, a trigger or a timebase circuit defect. The origin of the failure may be localized via the DDR? or the self test *TST? query.

(6) The Query Error bit (QYE) is set true (1) whenever (a) an attempt is made to read data from the Output Queue when no output is either present or pending, (b) data in the Output Queue has been lost, (c) both output and input buffers are full (deadlock state), (d) an attempt is made by the controller to read before having sent an <END>, (e) a command is received before the response to the previous query was read (output buffer flushed).

(7) The Request Control bit (RQC) is always false (0), as the oscilloscope has no GPIB controlling capability.

(8) The Operation Complete bit (OPC) is set true (1) whenever *OPC has been received, since commands and queries are strictly executed in sequential order. The oscilloscope starts processing a command only when the previous command has been entirely executed.
**STATUS**

**DESCRIPTION**

The EXR? query reads and clears the contents of the Execution error Register (EXR). The EXR register specifies the type of the last error detected during execution.

**QUERY SYNTAX**

EXR?

**RESPONSE FORMAT**

EXR <value>
<value>: = to

**EXAMPLE**

The following instruction reads the contents of the EXR register:

Command message:
EXR?

Response message (if no fault):
EXR 0

**RELATED COMMANDS**

ALL_STATUS, *CLS

**ADDITIONAL INFORMATION**

### Execution Error Status Register Structure (EXR)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Permission error. The command cannot be executed in local mode.</td>
</tr>
<tr>
<td>22</td>
<td>Environment error. The instrument is not configured to correctly process a command. For instance, the oscilloscope cannot be set to RIS at a slow timebase.</td>
</tr>
<tr>
<td>23</td>
<td>Option error. The command applies to an option which has not been installed.</td>
</tr>
<tr>
<td>25</td>
<td>Parameter error. Too many parameters specified.</td>
</tr>
<tr>
<td>26</td>
<td>Non-implemented command.</td>
</tr>
<tr>
<td>32</td>
<td>Waveform descriptor error. An invalid waveform descriptor has been detected.</td>
</tr>
<tr>
<td>36</td>
<td>Panel setup error. An invalid panel setup data block has been detected.</td>
</tr>
<tr>
<td>50</td>
<td>No mass storage present when user attempted to access it.</td>
</tr>
<tr>
<td>53</td>
<td>Mass storage was write protected when user attempted to create, or a file, to delete a file, or to format the device.</td>
</tr>
<tr>
<td>58</td>
<td>Mass storage file not found.</td>
</tr>
<tr>
<td>59</td>
<td>Requested directory not found.</td>
</tr>
<tr>
<td>61</td>
<td>Mass storage filename not DOS compatible, or illegal filename.</td>
</tr>
<tr>
<td>62</td>
<td>Cannot write on mass storage because filename already exists.</td>
</tr>
</tbody>
</table>
**FRAME_PARAM?, FPAR?**  
Query

**DESCRIPTION**  
The FRAME_PARAM command is used to get frame param include descriptor name, product name, the total number of frames and so on.

**QUERY SYNTAX**  
FPAR?

**Note:**  
This command is used with the history function.

**RESPONSE FORMAT**  
The format of the response is binary.

**EXAMPLE**  
The following command gets the frame parameters:

Command message:  
FPAR?
The FRAME_SET command is used to set history current frame number.

**COMMAND SYNTAX**

```
FRAM <frame_num>
```

Frame_num = 0 to the max frame number

**Note:** This command is used with the history function.

**EXAMPLE**

The following command sets current frame number to 50.

Command message:

```
FRAM 50
```
**DESCRIPTION**

The FRAME_TIME command is used to get current frame Acq. Time.

**QUERY SYNTAX**

FTIM?

**Note:** This command is used in history function opening.

**RESPONSE FORMAT**

The format of response is binary.

**EXAMPLE**

The following query reads the current frame Acq.Time.

Command message:

FTIM?
The FILTER command enables or disables filter of the specified trace.

The response to the FILTER? query indicates whether the filter of specified trace is enabled.

**COMMAND SYNTAX**

```
<channel>:FILTER <state>
<channel>: = {C1,C2,C3,C4}
<state>: = {ON,OFF}
```

**QUERY SYNTAX**

```
<channel>:FILTER?
```

**RESPONSE FORMAT**

```
<channel>:FILTER <state>
```

**EXAMPLE**

The following command enables the filter of channel 1:

Command message:

```
C1:FILT ON
```

**RELATED COMMANDS**

FILTS

**Note:**
This command is suitable for non-SPO models.
FUNCTION

DESCRIPTION

The FILT_SET command selects the specified type of filter, and sets the limit value of filter.

The response to the FILT_SET? query indicates current parameter of the filter.

COMMAND SYNTAX

<channel>:FILT_SET
TYPE,<type>,<limit>,<limit_value>

<channel>: = {C1,C2,C3,C4}
<type>: = {LP,HP,BP,BR}
<limit>: = {UPPLIMIT,LOWLIMIT}

Note:
1. LP is low-pass, HP is high-pass, BP is band-pass, BR is band-reject.
2. If seted the <limit>, the <type> must be related.

QUERY SYNTAX

<channel>: FILT_SET?

RESPONSE FORMAT

<channel>:FILTER TYPE,<type>,<limit>,<limit-value>

EXAMPLE

The following command changes the type of filter to band-pass, and sets the up-limit to 200 KHz and the low-limit to 100 KHz:

Command message:
C1:FILTS TYPE,BP,
UPPLIMIT,200KHz,LOWLIMIT,100KHz

RELATED COMMANDS

FILT

Note:
This command is suitable for non-SPO models.
**MATH**

**DESCRIPTION**
The FFT_WINDOW command selects the window of FFT (Fast Fourier Transform algorithm).

The response to the FFT_WINDOW? query indicates current window of FFT.

**COMMAND SYNTAX**

```
FFT_WINDOW <window>
```

<window>: = {RECT,BLAC,HANN,HAMM}

RECT is short for rectangle.
BLAC is short for Blackman.
HANN is short for hanning.
HAMM is short for hamming.

**QUERY SYNTAX**

```
FFT_WINDOW?
```

**RESPONSE FORMAT**

```
FFT_WINDOW,<window>
```

**EXAMPLE**
The following command sets the FFT window to hamming:

Command message:
FFTW HAMM
**MATH**

**FFT_ZOOM,FFTZ Command/Query**

**DESCRIPTION**

The FFT_ZOOM command selects the specified zoom of FFT.

The response to the FFT_ZOOM? query indicates current zoom in/out of FFT.

**COMMAND SYNTAX**

```
FFT_ZOOM <zoom>
< zoom >: = \{1,2,5,10\}
```

**QUERY SYNTAX**

```
FFT_ZOOM?
```

**RESPONSE FORMAT**

```
FFT_ZOOM,<zoom>
```

**EXAMPLE**

The following command sets the zoom factor of FFT to 1X:

Command message:

```
FFTZ 1
```

**Note:**

This command is suitable for the non-SPO models.
**DESCRIPTION**  
The FFT_SCALE command selects the specified scale of FFT (Fast Fourier Transform algorithm).

The response to the FFT_SCALE? query indicates current vertical scale of FFT waveform.

**COMMAND SYNTAX**  
```
FFT_SCALE <scale>
< scale >: = { 0.1, 0.2, 0.5, 1, 2, 5, 10, 20}
```

**QUERY SYNTAX**  
```
FFT_SCALE?
```

**RESPONSE FORMAT**  
```
FFT_SCALE < scale >
```

**EXAMPLE**  
The following command turns the vertical scale of FFT to 5dBVrms:

Command message:
```
FFTS 5
```
**DESCRIPTION**

The FFT_UNIT command selects the specified scale unit of FFT (Fast Fourier Transform algorithm).

The response to the FFT_UNIT? query indicates current vertical scale unit of FFT waveform.

**COMMAND SYNTAX**

<table>
<thead>
<tr>
<th>Command Syntax</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT_UNIT &lt;unit&gt;</td>
<td>Selects the specified scale unit of FFT.</td>
</tr>
<tr>
<td>&lt; unit &gt;: = {VRMS, DBVRMS}</td>
<td>Supported units.</td>
</tr>
</tbody>
</table>

**QUERY SYNTAX**

<table>
<thead>
<tr>
<th>Query Syntax</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT_UNIT?</td>
<td>Queries the current vertical scale unit of FFT waveform.</td>
</tr>
</tbody>
</table>

**RESPONSE FORMAT**

<table>
<thead>
<tr>
<th>Response Format</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT_UNIT,&lt; unit &gt;</td>
<td>Indicates the current vertical scale unit of FFT waveform.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

The following command turns the vertical scale unit of FFT to dBVrms:

Command message:

```
FFTS DBVRMS
```
**DESCRIPTION**

The FFT_TDIV command selects the horizontal scale of FFT (Fast Fourier Transform algorithm).

The response to the FFT_TDIV? query indicates current horizontal scale of FFT waveform.

**COMMAND SYNTAX**

```
FFT_TDIV <value>
< value > :
```

**QUERY SYNTAX**

```
FFT_TDIV?
```

**RESPONSE FORMAT**

```
FFT_TDIV < value >
```

**EXAMPLE**

The following command turns the vertical scale unit of FFT to 20MHz:

Command message:

```
FFTT 20MHz
```
**FFT_POSITION, FFTP**

**Command / Query**

**DESCRIPTION**

The FFT_POSITION command adjusts the vertical position of the FFT waveform on the screen. It does not affect the original offset value obtained at acquisition time.

The FFT_POSITION? query returns the current vertical position of the FFT waveform.

**COMMAND SYNTAX**

```
FFT_POSITION <display_offset>
```

Where:

- `<display_offset>`: 
  - `-20 DIV` to `20 DIV`
  - Note: The suffix DIV is optional.

**QUERY SYNTAX**

```
FFT_POSITION?
```

**RESPONSE FORMAT**

```
FFT_POSITION <display_offset>
```

**EXAMPLE**

The following shifts FFT waveform upwards by +3 divisions relative to the position at the time of acquisition:

Command message:
```
FFT_POSITION 3DIV
FFT_POSITION 3V (it assumes that the current vertical scale is 1V)
```

Note:
This command is suitable for the SPO models.
**DESCRIPTION**

The FFT_CENTER command selects the center frequency of FFT (Fast Fourier Transform algorithm).

The response to the FFT_CENTER? query indicates current center frequency of FFT waveform.

**COMMAND SYNTAX**

FFT_CENTER <value>
<value>:

**QUERY SYNTAX**

FFT_CENTER?

**RESPONSE FORMAT**

FFT_CENTER <value>

**EXAMPLE**

The following command sets the center frequency of FFT to 100MHz:

Command message:
FFTC 100MHz
**FFT_FULLSCREEN**, FFTFULLSCREEN

**Command / Query**

**DESCRIPTION**

The FFT_FULLSCREEN command enables or disables to display the FFT waveform full screen.

The response to the FFT_FULLSCREEN? query indicates whether the FFT waveform is full screen displayed.

**COMMAND SYNTAX**

FFT_FULLSCREEN <state>
<state>: = {ON,OFF}

**QUERY SYNTAX**

FFT_FULLSCREEN?

**RESPONSE FORMAT**

FFT_FULLSCREEN <state>

**EXAMPLE**

The following command enables to display the FFT waveform full screen:

Command message:
FFT ON
**DISPLAY**

**GRID_DISPLAY, GRDS**

**Description**
The GRID_DISPLAY command selects the type of the grid which is used to display.

The response to the GRID_DISPLAY? query indicates current type of the grid.

**Command Syntax**
GRID_DISPLAY <type>

<type> = {FULL, HALF, OFF}

**Query Syntax**
GRID_DISPLAY?

**Response Format**
GRID_DISPLAY <type>

**Example**
The following command changes the type of grid to full grid:

Command message:
GRID_DISPLAY FULL
**DESCRIPTION**

The HOR_MAGNIFY command horizontally expands the selected expansion trace by a specified factor. Magnification factors not within the range of permissible values will be rounded off to the closest legal value.

If the specified factor is too large for any of the expanded traces (depending on their current source), it is reduced to an acceptable value and only then applied to the traces. The VAB bit (bit 2) in the STB register is set when a factor outside the legal range is specified.

The HOR_MAGNIFY query returns the current magnification factor for the specified expansion function.

**COMMAND SYNTAX**

Format 1:
<exp_trace>: Hor_MAGnify <factor>
<exp_trace>: = {TA, TB, TC, TD}

<factor>: = 1 to 2,000,000 The range of <factor> is related to the current timebase and the range of the timebase.

Format 2:
Hor_MAGnify <value>

<value>:=
{1NS,2NS,5NS,10NS,20NS,50NS,100NS,200NS,500NS,1US,2US,5US,10US,20US,50US,100US,200US,500US,1MS,2MS,5MS,10MS,20MS}. The range of <value> is related to the current timebase and the range of the timebase.

**Note:**
Format 1 is suitable for non_SPO models. Format 2 is suitable for SPO models.

**QUERY SYNTAX**

Format 1:
<exp_trace>: Hor_MAGnify?
Format 2:
Hor_MAGnify?

RESPONSE FORMAT

<exp_trace>: Hor_MAGnify <factor>
Hor_MAGnify <value>

EXAMPLE

The following instruction horizontally magnifies Trace A (TA) by a factor of 5 for non_SPO models:
Command message:
TA: HMAG 5.00

The following instruction horizontally magnifies by value of 1US for SPO models:
Command message:
HMAG 1US

RELATED COMMANDS

HPOS
Digital Oscilloscopes Series

ZOOM

HOR_POSITION, HPOS
Command /Query

DESCRIPTION

The HOR_POSITION command horizontally positions the geometric center of the intensified zone on the source trace. Allowed positions range from division -7 to 7. If this would cause the horizontal position of any expanded trace to go outside the left or right screen boundaries, the difference of positions is adapted and then applied to the traces.

The VAB bit (bit 2) in the STB register is set if a value outside the legal range is specified.

The HOR_POSITION query returns the position of the geometric center of the intensified zone on the source trace.

COMMAND SYNTAX

Format 1:
<exp_trace>: Hor_POSition <hor_position>
<exp_trace>: = {TA, TB, TC, TD}
<hor_position>: = -7 to 7 DIV

Format 2:
Hor_POSition <hor_position>
<hor_position>: = -7 to 7 DIV

Note:
1. Format 1 is suitable for non_SPO models. It doesn’t distinguish expanded traces in Format 2 which is suitable for SPO models.
2. The range of the <hor_position> is related to the magnification factors of command HMAG. The range after magnifying which beyond the screen could display, and it will be adjusted to the proper value.
3. You need add the time unit to the hor_position when using the Format 2.

QUERY SYNTAX

Format 1:
<exp_trace>: Hor_POSition?

Format 2:
Hor_POSition?

RESPONSE FORMAT

<exp_trace>: Hor_POSition <hor_position>
Hor_POSition <hor_position>

EXAMPLE

The following instruction positions the center of the intensified zone on the trace currently viewed by Trace A (TA) at division 3:
Command message:
TA: HPOS 3

The following instruction positions the center of the intensified zone at 100NS:
Command message:
HPOS 100NS

RELATED COMMANDS

HMAG
**SYSTEM**

**DESCRIPTION**

The *IDN? query is used for identification purposes. The response consists of four different fields providing information on the manufacturer, the scope model, the serial number and the firmware revision level.

**QUERY SYNTAX**

*IDN?

**RESPONSE FORMAT**

*IDN \_SIGLENT, <model>, <serial_number>, <firmware_level>

- `<model>`: = A eleven characters model identifier
- `<serial_number>`: = A 14-digit decimal code
- `<firmware_level>`: = similar to k.xx.yy.zz

**EXAMPLE**

This example issues an identification request to the scope:

Command message:
*IDN?

Response message:
*IDN SIGLENT SDS1102CML,SDS00002110025, 3.01.01.22
**DESCRIPTION**

The INTENSITY command sets the intensity level of the grid or the trace.

The intensity level is expressed as a percentage (PCT). A level of 100 PCT corresponds to the maximum intensity whilst a level of 0 PCT sets the intensity to its minimum value. (The minimum value of the trace is 30 PCT)

The response to the INTENSITY? Query indicates the grid and trace intensity levels.

**COMMAND SYNTAX**

INTenSity GRID, <value>, TRACE, <value>

<value>: = 0(or 30) to 100 [PCT]

**Note:**
1. Parameters are grouped in pairs. The first of the pair names the variable to be modified, whilst the second gives the new value to be assigned. Pairs may be given in any order and be restricted to those variables to be changed.
2. The suffix PCT is optional.

**QUERY SYNTAX**

INTenSity?

**RESPONSE FORMAT**

INTenSity TRACE, <value>, GRID, <value>

**EXAMPLE**

The following instruction enables remote control of the intensity, and changes the grid intensity level to 75%:

Command message:
INTS GRID, 75
**DESCRIPTION**

The INR? query reads and clears the contents of the INternal state change Register (INR). The INR register (table below) records the completion of various internal operations and state transitions.

**Note:**
This command only supports 0 bit and 13 bit.

**QUERY SYNTAX**

INR?

**RESPONSE FORMAT**

INR <value>

<value>: = 0 to 65535

**EXAMPLE**

If we send INR? query after triggering the INR register:

Command message 1:
INR?

Response message 1:
INR 8193

If we send INR? query while the instrument hasn’t triggered vet(ARM), the INR register:

Command message 2:
INR?

Response message 2:
INR 8192

If we send INR? query after have sent a INR? query and the mode of the instrument is STOP the INR register:

Command message 3:
INR?

Response message 3:
INR 0
If we send INR? query while there is no trigger then trigger and finally send another INR? query the INR register:

Command message 4:
INR?

Response message 4:
INR 1

RELATED COMMANDS

ALL_STATUS?, *CLS

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>1</td>
<td>Trigger is ready</td>
</tr>
<tr>
<td>1</td>
<td>Pass/Fail test detected desired outcome</td>
</tr>
<tr>
<td>1</td>
<td>Waveform processing has terminated in Trace D</td>
</tr>
<tr>
<td>1</td>
<td>Waveform processing has terminated in Trace C</td>
</tr>
<tr>
<td>1</td>
<td>Waveform processing has terminated in Trace B</td>
</tr>
<tr>
<td>1</td>
<td>A memory card, floppy or hard disk exchange has been detected</td>
</tr>
<tr>
<td>1</td>
<td>Memory card, floppy or hard disk has become full in “AutoStore Fill” mode</td>
</tr>
<tr>
<td>0</td>
<td>Reserved for LeCroy use</td>
</tr>
<tr>
<td>1</td>
<td>A segment of a sequence waveform has been acquired</td>
</tr>
<tr>
<td>1</td>
<td>A time-out has occurred in a data block transfer</td>
</tr>
<tr>
<td>1</td>
<td>A return to the local state is detected</td>
</tr>
<tr>
<td>1</td>
<td>A screen dump has terminated</td>
</tr>
<tr>
<td>1</td>
<td>A new signal has been acquired</td>
</tr>
</tbody>
</table>
**CHANNEL**

**INVERTSET, INVS**

Command/Query

**DESCRIPTION**

The INVERTSET command inverts the specified traces or the math waveform.

The response to the INVERTSET? query indicates whether the specified waveform is inverted or not.

**COMMAND SYNTAX**

```
<trace>:INVERTSET < state >
<trace> := \{C1,C2,C3,C4,MATH\}
< state > := \{ON, OFF\}
```

**QUERY SYNTAX**

```
<trace>:INVERTSET?
```

**RESPONSE FORMAT**

```
<trace>:INVERTSET < state >
```

**EXAMPLE**

The following instruction inverts the trace of channel 1:

Command message:

C1:INVS ON
**DISPLAY**

**MENU, MENU**

**Command/Query**

**DESCRIPTION**

The MENU command enables or disables the display of the menu.

The response to the MENU? query indicates whether the menu is displayed.

**COMMAND SYNTAX**

```
MENU < state>
<state>::= {ON,OFF}
```

**QUERY SYNTAX**

```
MENU?
```

**RESPONSE FORMAT**

```
MENU < state>
```

**EXAMPLE**

The following instruction enables the display of the menu:

Command message:

```
MENU ON
```

**Note:**
This command is suitable for the model which has this function.
MATH

MATH_VERT_POS, MTVP

Command/Query

DESCRIPTION

The MATH_VERT_POS command controls the vertical position of the math waveform with specified source.

The FFT waveform isn’t included. But we have another command which called VPOS to control its vertical position.

The response to the MATH_VERT_POS? query indicates the value of the vertical position of the math waveform.

COMMAND SYNTAX

MATH_VERT_POS <position>

<position>::= the position is related to the position of the screen center. For example, if we set the position of MTVP to 50. The math waveform will be displayed 1 grid up to the vertical center of the screen. Namely one grid is 50.

QUERY SYNTAX

MATH_VERT_POS?

RESPONSE FORMAT

MATH_VERT_POS < position >

EXAMPLE

The following instruction changes the vertical position of the math waveform to 1 grid up to the screen vertical centre:

Command message:
MTVP 50
**MATH**

**MATH_VERT_DIV, MTVD**

Command/Query

**DESCRIPTION**

The MATH_VERT_DIV command controls the vertical sensitivity of the math waveform of the specified source. We can only set the value of existing math waveforms.

The FFT waveform isn’t included.

The response to the MATH_VERT_DIV? query indicates the specified scale of math waveform of specified source.

**COMMAND SYNTAX**

MATH_VERT_DIV < scale >

< scale > := 1PV/div ~ 100V/div.

**QUERY SYNTAX**

MATH_VERT_DIV?

**RESPONSE FORMAT**

MATH_VERT_DIV < scale >

**EXAMPLE**

The following instruction changes the vertical sensitivity of the math waveform of the specified source to 1V/div:

Command message:
MTVD 1V
**DESCRIPTION**

The MEMORY_SIZE command sets the maximum depth of memory.

The response to the MEMORY_SIZE? query the maximal depth of memory.

**COMMAND SYNTAX**

MEMORY_SIZE <size>

<size>:= {7K, 14K, 70K, 140K, 700K, 1.4M, 7M, 14M}

**QUERY SYNTAX**

MEMORY_SIZE?

**RESPONSE FORMAT**

MEMORY_SIZE <size>

**EXAMPLE**

The following instruction sets the maximum depth of memory to 14M.

Command message:

MSIZ 14M

**Note:**

This command is suitable for SPO models.
**MEASURE**

**DESCRIPTION**

The MEASURE_DELAY command selects the type of delay measure.

The response to the MEASURE_DELAY? query indicates the type of delay measure.

**COMMAND SYNTAX**

MEASURE_DELAY <type>,<source>

<source>:= \{C1-C2, C1-C3, C1-C4, C2-C3, C2-C4, C3-C4\}
<type>:=\{PHA,FRR,FRF,FFF,LRR,LRF,LFR,LFF,SKEW\}

The PHA is phase, the others are the same as the specified type of the instrument’s delay measure.

**QUERY SYNTAX**

<source>:MEAsure_Delay? <type>

**RESPONSE FORMAT**

<source>:MEAD <type>,<value>

**EXAMPLE**

The following instruction sets the type of delay measure to phase between C1 and C2.

Command message:
MEAD PHA,C1-C2
The OFFSET command allows adjustment of the vertical offset of the specified input channel. The maximum ranges depend on the fixed sensitivity setting.

If an out-of-range value is entered, the oscilloscope is set to the closest possible value and the VAB bit (bit 2) in the STB register is set.

The OFFSET? query returns the offset value of the specified channel.

**COMMAND SYNTAX**

```
<channel>: OFfSeT <offset>
<channel> = {C1, C2, C3,C4}
<offset> = See the oscilloscope’s specifications.
```

**QUERY SYNTAX**

```
<channel>: OFfSeT?
```

**RESPONSE FORMAT**

```
<channel>: OFfSeT <offset>
```

**EXAMPLE**

The following command sets the offset of Channel 2 to -3 V:

```
Command message:
C2:OFST -3V
```
### STATUS

#### *OPC*
Command/Query

### DESCRIPTION

The *OPC* (OPeration Complete) command sets the OPC bit (bit 0) in the standard Event Status Register (ESR). This command has no other effect on the operation of the oscilloscope because the instrument starts parsing a command or query only after it has completely processed the previous command or query.

The *OPC?* query always responds with the ASCII character “1” because the oscilloscope only responds to the query when the previous command has been entirely executed.

### COMMAND SYNTAX

*OPC*

### QUERY SYNTAX

*OPC?*

### RESPONSE FORMAT

*OPC 1*
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>The PARAMETER_CLR command clears the P/F test counter and starts it again at 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND SYNTAX</td>
<td>PARAMETER_CLR</td>
</tr>
<tr>
<td>RELATED COMMANDS</td>
<td>PARAMETER_VALUE PFDD</td>
</tr>
</tbody>
</table>
**MEASURE**

**PARAMETER_CUSTOM, PACU**

**Command**

**DESCRIPTION**

The PARAMETER_CUSTOM command controls the parameters that have customizable qualifiers.

**Note:**
The measured value of a parameter setup with PACU can be read by using PAVA?

**COMMAND SYNTAX**

```
PArarameter_CUstom <parameter>,<qualifier>

<parameter>: = {PKPK, MAX, MIN, AMPL, TOP,
BASE, CMEAN, MEAN, RMS, CRMS, OVSN,
FPRE, OVSP, RPRE, PER, FREQ, PWID, NWID,
RISE,FALL,WID,DUTY,NDUTY, ALL}
<qualifier>: = { C1,C2,C3,C4 }
```

Measurement qualifier specific to each(source option)

**EXAMPLE**

The following sets the type of measure to PKPK of Channel 1.

Command message:

```
PACU PKPK, C1
```

**RELATED COMMANDS**

PARAMETER_CLR, PARAMETER_VALUE
# Digital Oscilloscopes Series

## MEASURE

### PARAMETER_VALUE?, PAVA?

**Query**

### DESCRIPTION

The `PARAMETER_VALUE` query returns the measurement values.

### QUERY SYNTAX

```
<trace>:PARAMeter_Value? [<parameter>, ... , <parameter>]
```

- `<trace>`: = { C1, C2, C3, C4}
- `<parameter>`: = See table of parameter names on previous page.

### RESPONSE FORMAT

```
<trace>: PArmeter_VAlue <parameter>, <value> [, ... , <parameter>,<value>]
```

### EXAMPLE

The following query reads the rise time of Channel 2

- **Command message:**
  - C2:PAVA? RISE

- **Response message:**
  - C2:PAVA RISE, 3.6E-9S

### RELATED COMMANDS

CURSOR_MEASURE, CURSOR_SET, PARAMETER_CUSTOM

See the table on the following page for all of the parameters:
### Parameters Available on All Models

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>all parameters</td>
<td>NDUTY</td>
<td>negative duty cycle</td>
</tr>
<tr>
<td>AMPL</td>
<td>amplitude</td>
<td>NWID</td>
<td>negative width</td>
</tr>
<tr>
<td>BASE</td>
<td>base</td>
<td>OVSN</td>
<td>negative overshoot</td>
</tr>
<tr>
<td>CMEAN</td>
<td>mean for cyclic waveform</td>
<td>OVSP</td>
<td>positive overshoot</td>
</tr>
<tr>
<td>CRMS</td>
<td>root mean square for cyclic part of waveform</td>
<td>PKPK</td>
<td>peak-to-peak</td>
</tr>
<tr>
<td>DUTY</td>
<td>duty cycle</td>
<td>PER</td>
<td>period</td>
</tr>
<tr>
<td>FALL</td>
<td>falltime</td>
<td>RPRE</td>
<td>(Vmin-Vbase)/ Vamp before the waveform rising transition</td>
</tr>
<tr>
<td>FREQ</td>
<td>frequency</td>
<td>PWID</td>
<td>positive width</td>
</tr>
<tr>
<td>FPRE</td>
<td>(Vmin-Vbase)/ Vamp before the waveform falling transition</td>
<td>RMS</td>
<td>root mean square</td>
</tr>
<tr>
<td>MAX</td>
<td>maximum</td>
<td>RISE</td>
<td>risetime</td>
</tr>
<tr>
<td>MIN</td>
<td>minimum</td>
<td>TOP</td>
<td>top</td>
</tr>
<tr>
<td>MEAN</td>
<td>mean</td>
<td>WID</td>
<td>width</td>
</tr>
</tbody>
</table>

#### Custom Parameters Defined using PARAMETER_CUSTOM Command

<table>
<thead>
<tr>
<th>Custom Parameter</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST1</td>
<td></td>
</tr>
<tr>
<td>CUST2</td>
<td></td>
</tr>
<tr>
<td>CUST3</td>
<td></td>
</tr>
<tr>
<td>CUST4</td>
<td></td>
</tr>
<tr>
<td>CUST5</td>
<td></td>
</tr>
</tbody>
</table>
**DISPLAY**

**PERSIST_SETUP, PESU**

Command /Query

**DESCRIPTION**

The PERSIST_SETUP command selects the persistence duration of the display, in seconds, in persistence mode.

The PERSIST_SETUP? query indicates the current status of the persistence.

**COMMAND SYNTAX**

PErsist_SetUp <time>

<time> := {1, 5, 10, 30, Infinite, OFF}

**Note:**
The options of time are the same as your oscilloscope.

**QUERY SYNTAX**

PErsist_SetUp?

**RESPONSE FORMAT**

PErsist_SetUp <time>

**EXAMPLE**

The following instruction sets the variable persistence at 5 seconds:

Command message:

PESU 5

**RELATED COMMANDS**

PERSIST
SAVE/RECALL

PANEL_SETUP, PNSU
Command/Query

DESCRIPTION
The PANEL_SETUP command complements the *SAV or *RST commands. PANEL_SETUP allows you to archive panel setups in encoded form on external storage media. Only setup data read by the PNSU? query can be recalled.

COMMAND SYNTAX
PaNel_SetUp <setup>
<setup>: = A setup previously read by PNSU?

QUERY SYNTAX
PaNel_SetUp?

RESPONSE FORMAT
PaNel_SetUp <setup>

EXAMPLE
The following instruction saves the oscilloscope’s current panel setup in the file PANEL.SET:

Command message:
PNSU

RELATED COMMANDS
*RCL, *SAV
**PASS/FAIL**

**PF_DISPLAY,PFDS**

Command /Query

**DESCRIPTION**

The PF_DISPLAY command enables or disables to turn the test and display the message in the pass/fail option.

The response to the PF_DISPLAY? query indicates whether the test is enabled and the message of pass/fail is displayed.

**COMMAND SYNTAX**

PF_DISPLAY TEST,<state>,DISPLAY,<state>

<state> = {ON, OFF}

**QUERY SYNTAX**

PF_DISPLAY?

**RESPONSE FORMAT**

PF_DISPLAY TEST <state>,DISPLAY,<state>

**EXAMPLE**

The following instruction enables to turn on the test and display the message of pass/fail:

Command message:
PFDS TEST,ON,DISPLAY,ON
PASS/FAIL

PF_SAVELOAD, PFSL
Command

DESCRIPTION
The PF_SAVELOAD command saves or recalls the created mask setting.

COMMAND SYNTAX
PF_SAVELOAD
LOCATION,<location>,ACTION,<action>
The <location> means to save the created mask setting to the internal memories or the external memories.

<location> := \{IN,EX\}
IN means to save the mask setting to the internal memories while EX means the external memories.

<action> := \{SAVE,LOAD\}
SAVE means to save the mask setting while LOAD means recall the stored mask setting.

EXAMPLE
The following instruction saves the mask setting to the internal memories:

Command message:
PFSL LOCATION,IN,ACTION,SAVE

RELATED COMMANDS
PFCM

Note:
This command is suitable for non-SPO models.
**PASS/FAIL**

**PF_SET,PFST**

Command /Query

**DESCRIPTION**

The PF_SET command sets the X mask and the Y mask of the mask setting in the pass/fail option.

The response to the PF_SET? query indicates the value of the X mask and the Y mask.

**COMMAND SYNTAX**

```
PF_SET XMASK, <div>, YMASK, <div>
```

<div> = 0.04div~4.0div

**QUERY SYNTAX**

```
PF_SET?
```

**RESPONSE FORMAT**

```
PF_SET XMASK, <div>, YMASK, <div>
```

**EXAMPLE**

The following instruction sets the X mask to 0.4div and the Y mask to 0.5div of the mask setting in the pass/fail option:

Command message:
```
PFST XMASK,0.4,YMASK,0.5
```

**RELATED COMMANDS**

PFSL
PASS/FAIL

DESCRIPTION
The PF_CONTROL command controls the pass/fail controlling options: “operate”, “output” and the “stop on output”.

See instrument’s Operator Manual for these options.

The response to the PF_CONTROL? query indicates the controlling options of the pass/fail.

COMMAND SYNTAX
PF_CONTROL
TRACE,<trace>,CONTROL,<control>,OUTPUT,<output>,OUTPUTSTOP,<state>

<trace> := {C1,C2,C3,C4}
<control> := {START,STOP}
<output> := {FAIL,PASS}
<state> := {ON,OFF}

QUERY SYNTAX
PF_CONTROL?

RESPONSE FORMAT
PF_CONTROL
TRACE,<trace>,CONTROL,<control>,OUTPUT,<output>,OUTPUTSTOP,<state>

EXAMPLE
The following instruction sets source to channel 1, “operate” to “start”, “output” to “pass” and “stop on output” to “off”:

Command message:
PFCT TRACE,C1,CONTROL,START,
OUTPUT,PASS,OUTPUTSTOP,OFF

RELATED COMMANDS
PF_SOURCE, PF_BUFFER, PF_OPERATION, PF_FAIL_STOP

Note:
This command is suitable for non-SPO models. For SPO models, please refer to related commands.
**DESCRIPTION**
The PF_SOURCE command controls the source of pass/fail.

The response to the PF_SOURCE? query indicates the source of the pass/fail.

**COMMAND SYNTAX**
PF_SOURCE <trace>

<trace> = {C1,C2,C3,C4}

**QUERY SYNTAX**
PF_SOURCE?

**RESPONSE FORMAT**
PF_SOURCE <trace>

**EXAMPLE**
The following instruction sets source to channel 1.

Command message:
PF_SOURCE C1

**Note:**
This command is suitable for SPO models.
**PF_BUFFER, PFBF**  
Command/Query

**DESCRIPTION**  
The PF_BUFFER command controls the pass/fail controlling option: “output”.  
See instrument’s Operator Manual for the option.  
The response to the PF_BUFFER? query indicates the controlling option “output” of the pass/fail.

**COMMAND SYNTAX**  
PF_BUFFER <state>  

<state>: = {ON, OFF}

**QUERY SYNTAX**  
PF_BUFFER?

**RESPONSE FORMAT**  
PF_BUFFER <state>

**EXAMPLE**  
The following instruction sets “output” to “ON”.  

Command message:  
PFBF ON

**Note:**  
This command is suitable for SPO models.
**PASS/FAIL**

**DESCRIPTION**

The PF OPERATION command controls the pass/fail controlling option: “operate”.

See instrument’s Operator Manual for the option.

The response to the PF OPERATION? query indicates the controlling option of the pass/fail.

**COMMAND SYNTAX**

PF OPERATION <control>

<control>: = {ON, OFF}

**QUERY SYNTAX**

PF OPERATION?

**RESPONSE FORMAT**

PF OPERATION <control>

**EXAMPLE**

The following instruction sets “operate” to “ON”.

Command message:
PFOP ON

**Note:**
This command is suitable for SPO models.
PASS/FAIL

PF_FAIL_STOP,PFFS
Command/Query

DESCRIPTION
The PF_FAIL_STOP command controls the pass/fail controlling option: “stop on fail”.

See instrument’s Operator Manual for the option.

The response to the PF_FAIL_STOP? query indicates the controlling option of the pass/fail.

COMMAND SYNTAX
PF_FAIL_STOP <state>

<state>: = {ON,OFF}

QUERY SYNTAX
PF_FAIL_STOP?

RESPONSE FORMAT
PF_FAIL_STOP <state>

EXAMPLE
The following instruction sets “stop on fail” to “off”.

Command message:
PFFS OFF

Note:
This command is suitable for SPO models.
**PASS/FAIL**

**PF_CREATEM,PFCM**  
Command

**DESCRIPTION**  
The PF_CREATEM command creates the mask of the pass/fail.

**COMMAND SYNTAX**  
PF_CREATEM

**EXAMPLE**  
The following instruction creates the mask of the pass/fail.

Command message:  
PFCM

**RELATED COMMANDS**  
PFSL,PFST
**PASS/FAIL**

**PF_DATADIS, PFDD**

*Query*

**DESCRIPTION**

The PF_DATADIS? query returns the number of the fail, pass and total number that the screen showing.

**COMMAND SYNTAX**

PF_DATADIS?

**RESPONSE FORMAT**

PF_DATADIS

FAIL,<num>,PASS,<num>,total,<num>

**EXAMPLE**

The following instruction returns the number of the message display of the pass/fail:

Command message:

PFDD FAIL,0,PASS,0,TOTAL,0
**SAVE/RECALL**

**DESCRIPTION**

The *RCL command sets the state of the instrument, using one of the ten non-volatile panel setups, by recalling the complete front-panel setup of the instrument. Panel setup 0 corresponds to the default panel setup.

The *RCL command produces the opposite effect of the *SAV command.

If the desired panel setup is not acceptable, the EXecution error status Register (EXR) is set and the EXE bit of the standard Event Status Register (ESR) is set.

**COMMAND SYNTAX**

*RCL <panel_setup>

<panel_setup>:= 0 to 20

**EXAMPLE**

The following recalls the instrument setup previously stored in panel setup 3:

Command message:

*RCL 3

**RELATED COMMANDS**

PANEL_SETUP, *SAV, EXR
The RECALL command recalls a waveform file from the current directory on mass storage into any or all of the internal memories M1 to M10 (or M20 in the CFL series).

COMMAND SYNTAX

<memory>: RECall DISK, <device>, FILE, ‘<filename>’
<memory>: = {M1~M10} (or M1~M20 in the CFL series)
<device>: = {UDSK}
<filename>: = A waveform file under a legal DOS path. A filename-string of up to eight characters, with the extension “.DAV”. (This can include the ‘/’ character to define the root directory.)

EXAMPLE

The following recalls a waveform file called “C1WF.DAV” from the memory card into Memory M1:

Command message:
M1:REC DISK, UDSK FILE, ‘C1WF.DAV’

RELATED COMMANDS

STORE, INR?

Note:
This command is suitable for non-SPO models.
**SAVE/RECALL**

**RECALL_PANEL, RCPN**

**Command**

**DESCRIPTION**

The RECALL_PANEL command recalls a front-panel setup from the current directory on mass storage.

**COMMAND SYNTAX**

ReCall_PaNel DISK, <device>, FILE, ‘<filename>’
<device> := {UDSK}
<filename> := A waveform file under a legal DOS path. A filename-string of up to eight characters, with the file extension. (For SDS1000X-E series, the ‘/’ character to define the root directory is not supported. And the file extension is “.XML”. For the other serials, the ‘/’ character to define the root directory is supported. And the file extension is “.SET”)

**EXAMPLE**

The following recalls the front-panel setup from file SEAN. SET in a USB memory device:

Command message:
RCPN DISK, UDSK, FILE, ‘SEAN. SET’

**RELATED COMMANDS**

PANEL_SETUP, *SAV, STORE_PANEL, *RCL
**SAVE/RECALL**

**DESCRIPTION**

The *RST command initiates a device reset. The *RST sets recalls the default setup.

**COMMAND SYNTAX**

*RST

**EXAMPLE**

This example resets the oscilloscope:

Command message:

*RST

**RELATED COMMANDS**

*CAL, *RCL
FUNCTION

REF_CLOSE, REFCL
Command

DESCRIPTION
The REF_CLOSE command closes the reference function.

COMMAND SYNTAX
REF_CLOSE

EXAMPLE
Command message:
REF_POSITION 0.2

Note:
This command is only supported by SDS1000X-E series.
**FUNCTION**

**REF_DISPLAY, REFDI**

Command / Query

**DESCRIPTION**

The REF_DISPLAY command enable or disable the current reference channel show on the screen.

**COMMAND SYNTAX**

REF_DISPLAY < state >
<source> := {ON, OFF}

Only used when the current reference channel that has stored waveform, and the reference function is enable.

**QUERY SYNTAX**

REF_DISPLAY?

**RESPONSE FORMAT**

REF_DISPLAY <state>

**EXAMPLE**

The following instruction enable the current reference channel:

Command message:
REF_DISPLAY ON,

**Note:**

This command is only supported by SDS1000X-E series.
FUNCTION

REF_LOCATION, REF_L

Command / Query

DESCRIPTION

The REF_LOCATION command sets the current reference channel.

The response to the REF_LOCATION? query return the current reference channel.

COMMAND SYNTAX

REF_LOCATION <location>
< location >:= {REFA,REFB}

Only used when the reference function is enable.

QUERY SYNTAX

REF_LOCATION?

RESPONSE FORMAT

REF_LOCATION <location>

EXAMPLE

The following instruction select REFA as the current reference channel:

Command message:
REF_LOCATION REFA

Note:
This command is only supported by SDS1000X-E series.
**FUNCTION**

**REF_POSITION, REFPO**

Command / Query

**DESCRIPTION**

The REF_POSITION command sets the vertical offset of the current reference channel.

The response to the REF_POSITION? query return the vertical offset of the current reference channel.

**COMMAND SYNTAX**

REF_POSITION < value >

Only used when the current reference channel that has stored waveform, and the display state is on.

**QUERY SYNTAX**

REF_POSITION?

**RESPONSE FORMAT**

REF_POSITION < value >

**EXAMPLE**

The following instruction sets the current reference channel vertical offset to 0.2V:

Command message:
REF_POSITION 0.2

**Note:**
This command is only supported by SDS1000X-E series.
**FUNCTION**

**REF_SAVE, REFSA Command**

**DESCRIPTION**

The REF_SAVE command saves the waveform source to the current reference channel.

**COMMAND SYNTAX**

REF_SAVE

**EXAMPLE**

Command message:

REF_SAVE

**Note:**

This command is only supported by SDS1000X-E series.
### DESCRIPTION

The `REF_SCALE` command sets the vertical scale of the current reference channel.

The response to the `REF_SCALE?` query return the vertical scale of the current reference channel.

### COMMAND SYNTAX

`REF_SCALE < value >`

Only used when the current reference channel that has stored waveform, and the display state is on.

### QUERY SYNTAX

`REF_SCALE?`

### RESPONSE FORMAT

`REF_SCALE < value >`

### EXAMPLE

The following instruction sets the current reference channel vertical scale to 0.1V:

Command message:

`REF_SCALE 0.1,`

### Note:

This command is only supported by SDS1000X-E series.
**FUNCTION**

**REF_SOURCE, REFSR**  
Command / Query

**DESCRIPTION**  
The REF_SOURCE command sets the reference waveform source.

The response to the REF_SOURCE? query return the source of the current reference channel.

**COMMAND SYNTAX**  
REF_SOURCE <source>  
<source> := {C1,C2,C3,C4,MATH}

Make sure choose channels currently enabled as the source.

**QUERY SYNTAX**  
REF_SOURCE?

**RESPONSE FORMAT**  
REF_SOURCE <source>

**EXAMPLE**  
The following instruction select C1 as the source of current reference channel:

Command message:  
REF_SOURCE C1

**Note:**  
This command is only supported by SDS1000X-E series.
FUNCTION

DESCRIPTION
The REF_SET command sets the reference waveform and its options.

The response to the REF_SET? query indicates whether the specified reference waveform is turned on.

COMMAND SYNTAX
REF_SET TRACE,<trace>REF,<ref>,state, <state>,SAVE,DO
<trace> := {C1,C2,C3,C4,MATH}
<ref> := {RA,RB,RC,RD}
The Rx(x is A,B,C,D) is that which one can be stored or displayed
<state> := {ON,OFF}
The state enables or disables to display the specified reference waveform.
If the command syntax include ‘SAVE,DO’, the specified trace will be saved to the specified reference waveform.

QUERY SYNTAX
REF_SET? REF,<ref>

RESPONSE FORMAT
REF_SET REF,<ref>,STATE,<state>

EXAMPLE
The following instruction saves the channel 1 waveform to the REFA, and turns on REFA:

Command message:
REFS TRACE,C1,REF,RA,
STATE,ON,SAVE,DO

Note:
This command is not supported by SDS1000X-E series.
**SAVE/RECALL**

### *SAV Command*

**DESCRIPTION**

The *SAV command stores the current state of the instrument in internal memory. The *SAV command stores the complete front-panel setup of the instrument at the time the command is issued.

**COMMAND SYNTAX**

*SAV <panel_setup>

<p PANEL_FORMAT:='text-align: right'> <panel_setup>: = 1 to 20</p>

**EXAMPLE**

The following saves the current instrument setup in Panel Setup 3:

Command message:

*SAV 3

**RELATED COMMANDS**

PANEL_SET UP, *RCL
**FUNCTION**  
SCREEN_DUMP,SCDP  
Command

**DESCRIPTION**  
The SCREEN_DUMP command is used to obtain the screen information of image format.

**COMMAND SYNTAX**  
SCreen_DumP

**EXAMPLE**  
The following command transfers the screen information of image format to the controller

Command message:  
SCDP
DISPLAY

DESCRIPTION The SCREEN_SAVE command controls the automatic Screen Saver, which automatically shuts down the internal color monitor after a preset time.

The response to the SCREEN_SAVE? query indicates whether the automatic screen saver feature is on or off.

Note:
1. When the screen save is in effect, the oscilloscope is still fully functional.

COMMAND SYNTAX
SCreen_SaVe <enabled>
<enabled>: = {YES, NO}
Note: the parameters for SDS1000X-E series are as follows:
<enabled>: = {OFF, 1MIN, 5MIN, 10MIN, 30MIN, 60MIN}

QUERY SYNTAX
SCreen_SaVe?

RESPONSE FORMAT
SCreen_SaVe <enabled>

EXAMPLE The following enables the automatic screen saver:

Command message:
SCSV YES
**STATUS**

**DESCRIPTION**

The *SRE command sets the Service Request Enable register (SRE). This command allows the user to specify which summary message bit(s) in the STB register will generate a service request.

A summary message bit is enabled by writing a ‘1’ into the corresponding bit location. Conversely, writing a ‘0’ into a given bit location prevents the associated event from generating a service request (SRQ). Clearing the SRE register disables SRQ interrupts.

The *SRE? query returns a value that, when converted to a binary number, represents the bit settings of the SRE register.

Note:
1. That bit 6 (MSS) cannot be set and its returned value is always zero.

**COMMAND SYNTAX**

*SRE <value>

<value> := 0 to 255

**QUERY SYNTAX**

*SRE?

**RESPONSE FORMAT**

*SRE <value>

**EXAMPLE**

The following instruction allows an SRQ to be generated as soon as the MAV summary bit (bit 4, i.e. decimal 16) or the INB summary bit (bit 0, i.e. decimal 1) in the STB register, or both, are set. Summing these two values yields the SRE mask 16+1 = 17.

Command message:
*SRE 17
**STATUS**

**DESCRIPTION**
The *STB? query reads the contents of the 488.1 defined status register (STB), and the Master Summary Status (MSS). The response represents the values of bits 0 to 5 and 7 of the Status Byte register and the MSS summary message.

The response to a *STB? Query is identical to the response of a serial poll except that the MSS summary message appears in bit 6 in place of the RQS message.

**QUERY SYNTAX**

*STB?

**RESPONSE FORMAT**

*STB <value>

<value>: = 0 to 255

**EXAMPLE**
The following reads the status byte register:

Command message:
*STB?

Response message:
*STB 0

**RELATED COMMANDS**
ALL_STATUS,*CLS,*SRE
### ADDITIONAL INFORMATION

#### Status Byte Register (STB)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Value</th>
<th>Bit Name</th>
<th>Description</th>
<th>Not</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>128</td>
<td>DIO7</td>
<td>0 reserved for future use</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>MSS/RQS</td>
<td>at least 1 bit in STB masked by SRE is 1 service is requested</td>
<td>(1) (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSS=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RQS=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>ESB</td>
<td>1 an ESR enabled event has occurred</td>
<td>(3)</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>MAV</td>
<td>1 output queue is not empty</td>
<td>(4)</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>DIO3</td>
<td>0 reserved</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>VAB</td>
<td>1 a command data value has been adapted</td>
<td>(5)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>DIO1</td>
<td>0 reserved</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>INB</td>
<td>1 an enabled INternal state change has occurred</td>
<td>(6)</td>
</tr>
</tbody>
</table>

#### Notes

1. The Master Summary Status (MSS) indicates that the instrument requests service, whilst the Service Request status — when set — specifies that the oscilloscope issued a service request. Bit position 6 depends on the polling method:
   - Bit 6 = MSS if an *STB? Query is received
   - = RQS if serial polling is conducted

2. Example: If SRE=10 and STB=10 then MSS=1. If SRE=010 and STB=100 then MSS=0.

3. The Event Status Bit (ESB) indicates whether or not one or more of the enabled IEEE 488.2 events have occurred since the last reading or clearing of the Standard Event Status Register (ESR). ESB is set if an enabled event becomes true (1).

4. The Message AVailable bit (MAV) indicates whether or not the Output queue is empty. The MAV summary bit is set true (1) whenever a data byte resides in the Output queue.

5. The Value Adapted Bit (VAB) is set true (1) whenever a data value in a command has been adapted to the nearest legal value. For instance, the VAB bit would be set if the timebase is redefined as 2 μs/div since the adapted value is 2.5 μs/div.

6. The INternal state Bit (INB) is set true (1) whenever certain enabled internal states are entered. For further information, refer to the INR query.
### STOP Command

#### DESCRIPTION
The STOP command immediately stops the acquisition of a signal. If the trigger mode is AUTO or NORM.

#### QUERY SYNTAX
STOP

#### EXAMPLE
The following stops the acquisition process:

- **Command message:**
  ```
  *STOP
  ```
- **Response message:**
  ```
  *STB 0
  ```

#### RELATED COMMANDS
ARM_ACQUISITION, TRIG_MODE, WAIT
**SAVE/RECALL**

**STORE_PANEL, STPN**

Command

**DESCRIPTION**
The STORE_PANEL command stores the complete front-panel setup of the instrument, at the time the command is issued, into a file on the specified-DOS path directory in a USB memory device.

**COMMAND SYNTAX**

```
STore_PaNel DISK, <device>, FILE, ‘<filename>’
<device>: ={UDSK}
< directory >: =A legal DOS path or filename. A filename -string of up to 8 characters, with the file extension. (For SDS1000X-E series, the ‘/’ character to define the root directory is not supported. And the file extension is ‘.XML’. For the other serials, the ‘/’ character to define the root directory is supported. And the file extension is ‘.SET’.)
```

**EXAMPLE**
The following code saves the current instrument setup to root directory of the USB memory device in a file called “SEAN.SET”:

Command message:
```
STore_PaNel DISK, UDSK, FILE, ‘SEAN.SET’
```

The following code saves the current instrument setup to specified-directory of the USB memory device in a file called “SEAN.SET”:

Command message:
```
STore_PaNel DISK, UDSK, FILE, ‘/AAA/SEAN’
```

**RELATED COMMANDS**

*S, RECALL_PANEL, *RCL
**ACQUISITION**

**SAMPLE_STATUS, SAST**

Query

**DESCRIPTION**

The SAST? query the acquisition status of the scope.

**QUERY SYNTAX**

SAST?

**RESPONSE FORMAT**

SAST < status >

**EXAMPLE**

The following command reads the acquisition status of the scope.

Command message:
SAST?

Response message:
SAST trig’d
**DESCRIPTION**

The SARA? query returns the sample rate of the scope.

**QUERY SYNTAX**

SARA?

**RESPONSE FORMAT**

SARA< value >

**EXAMPLE**

The following command reads the sample rate of the scope.

Command message:
SARA?

Response message:
SARA 500.0kSa/s
**ACQUISITION**

**SAMPLE_NUM? ,SANU?**

**Query**

**DESCRIPTION**

The SANU? query returns the number of sampled points available from last acquisition and the trigger position.

**QUERY SYNTAX**

SANU? <channel>

**RESPONSE FORMAT**

SANU <value>

**EXAMPLE**

The following command reads the number of sampled points available from last acquisition from the Channel 2.

Command message:
SANU? C2

Response message:
SANU 700kpts
**DESCRIPTION**

The SET50 command sets the trigger level of the specified trigger source to the centre of the signal amplitude.

**COMMAND SYNTAX**

SET50

**EXAMPLE**

The following command sets the trigger level of the specified trigger source to the centre of the signal amplitude

Command message:

SET50
**ACQUISITION**

**SKEW,SKEW Command**

**DESCRIPTION**
The SKEW command sets the skew value of the specified trace.

The response to the SKEW? query indicates the skew value of the specified trace.

**COMMAND SYNTAX**
<trace>:SKEW <skew>
<trace>: = {C1,C2,C3,C4}
<skew>: = it is a value about time.

**QUERY SYNTAX**
<trace>:SKEW?

**RESPONSE FORMAT**
<trace>:SKEW <skew>

**EXAMPLE**
The following command sets channel 1 skew value to 3ns

Command message:
C1:SKEW 3NS
**DESCRIPTION**

The SINXX_SAMPLE command sets the way of interpolation.

The response to the SINXX_SAMPLE? query indicates the way of interpolation.

**COMMAND SYNTAX**

SINXX_SAMPLE, <state>

<state> : = {ON,OFF}

ON means sine interpolation, and OFF means linear interpolation.

**QUERY SYNTAX**

SINXX_SAMPLE?

**RESPONSE FORMAT**

SINXX_SAMPLE <state>

**EXAMPLE**

The following instruction sets the way of the interpolation to sine interpolation:

Command message:

SXSA ON
**ACQUISITION**

**TIME_DIV, TDIV**

**Command/Query**

**DESCRIPTION**

The TIME_DIV command modifies the timebase setting. The new timebase setting may be specified with suffixes: NS for nanoseconds, US for microseconds, MS for milliseconds, S for seconds, or KS for kiloseconds. An out-of-range value causes the VAB bit (bit 2) in the STB register to be set.

The TIME_DIV? query returns the current timebase setting.

**COMMAND SYNTAX**

Time_DIV <value>

<value>::={1NS,2NS,5NS,10NS,20NS,50NS,100NS,200NS,500NS,1US,2US,5US,10US,20US,50US,100US,200US,500US,1MS,2MS,5MS,10MS,20MS,50MS,100MS,200MS,500MS,1S,2S,5S,10S,20S,50S}

**QUERY SYNTAX**

Time_DIV?

**RESPONSE FORMAT**

Time_DIV <value>

**EXAMPLE**

The following sets the time base to 500 μs /div:

Command message:

TDIV 500US

**RELATED COMMANDS**

TRIG_DELAY, TRIG_MODE
**DESCRIPTION**

The TEMPLATE? query produces a copy of the template that describes the various logical entities making up a complete waveform. In particular, the template describes in full detail the variables contained in the descriptor part of a waveform.

**QUERY SYNTAX**

TeMPLate?

**RESPONSE FORMAT**

TeMPLate “<template>”

<template>: = A variable length string detailing the structure of a waveform.

**RELATED COMMANDS**

WF
**DESCRIPTION**

The TRACE command enables or disables the display of a trace. An environment error is set if an attempt is made to display more than four waveforms.

The TRACE? query indicates whether the specified trace is displayed or not.

**COMMAND SYNTAX**

```plaintext
<trace>: TRAc<mode>
<trace> = {C1, C2, C3, C4, TA, TB, TC, TD}
<mode> = {ON, OFF}
```

**QUERY SYNTAX**

```plaintext
<trace>: TRA?
```

**RESPONSE FORMAT**

```plaintext
<trace>: TRAc<mode>
```

**EXAMPLE**

The following command displays Channel 1(C1):

Command message:

C1:TRA ON
**TRIGGER**

**TRIG_COUPLING, TRCP**

**Command/Query**

**DESCRIPTION**

The TRIG_COUPLING command sets the coupling mode of the specified trigger source.

The TRIG_COUPLING? query returns the trigger coupling of the selected source.

**COMMAND SYNTAX**

<trig_source>: TRig_CouPling <trig_coupling>

<trig_source>: {C1, C2, C3, C4, EX, EX5, LINE}

<trig_coupling>: {AC, DC, HFREJ, LFREJ}

**QUERY SYNTAX**

<trig_source>: TRig_CouPling?

**RESPONSE FORMAT**

<trig_source>: TRig_CouPling <trig_coupling>

**EXAMPLE**

The following command sets the coupling mode of the trigger source Channel 2 to AC:

Command message:

C2: TRCP AC

**RELATED COMMANDS**

TRIG_COUPLING, TRIG_DELAY, TRIG_LEVEL, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE
The TRIG_DELAY command sets the time at which the trigger is to occur with respect to the first acquired data point.

This mode is called pre-trigger acquisition, as data are acquired before the trigger occurs. Negative trigger delays must be given in seconds. This mode is called post-trigger acquisition, as the data are acquired after the trigger has occurred.

If a value outside the range, the trigger time will be set to the nearest limit and the VAB bit (bit 2) will be set in the STB register. The response to the TRIG_DELAY? query indicates the trigger time with respect to the first acquired data point.

**COMMAND SYNTAX**

TRig_Delay <value>

<value>: the range of value is related to the timebase.

**Note:**
The suffix S is optional and assumed.

**QUERY SYNTAX**

TRig_Delay?

**RESPONSE FORMAT**

TRig_Delay <value>

**EXAMPLE**

The following command sets the trigger delay to -2ms (posttrigger):

Command message:

TRDL -2MS

**RELATED COMMANDS**

TIME_DIV, TRIG_COUPLING, TRIG_LEVEL, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE
The TRIG_LEVEL command adjusts the trigger level of the specified trigger source. An out-of-range value will be adjusted to the closest legal value and will cause the VAB bit (bit 2) in the STB register to be set.

The TRIG_LEVEL? query returns the current trigger level.

**COMMAND SYNTAX**

```
<trig_source>: TRig_LeVel <trig_level>
<trig_source>:= {C1, C2, C3, C4, EX, EX5}
<trig_level>:= -4.5DIV* volt/div to 4.5DIV * volt/div
```

*Note:* The suffix V is optional and assumed.

**QUERY SYNTAX**

```
<trig_source>:TRig_LeVel?
```

**RESPONSE FORMAT**

```
<trig_source>:TRig_LeVel <trig_level>
```

**EXAMPLE**

The following code adjusts the trigger level of Channel 3 to 52.00mv:

Command message:

```
C3:TRig_LeVel 52.00mv
```

**RELATED COMMANDS**

TRIG_COUPLING, TRIG_DELAY, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE
**DESCRIPTION**

The TRIG_LEVEL2 command adjusts the second trigger level of the specified trigger source. An out-of-range value will be adjusted to the closest legal value and will cause the VAB bit (bit 2) in the STB register to be set.

The TRIG_LEVEL? query returns the current trigger level.

**COMMAND SYNTAX**

\(<\text{trig\_source}>\): TRig_LeVel2 \(<\text{trig\_level}>\)

\(<\text{trig\_source}>\): = \{C1, C2, C3, C4\}

\(<\text{trig\_level}>\): = -4.5DIV * volt/div to 4.5DIV * volt/div

Note:  
The suffix V is optional and assumed.

**QUERY SYNTAX**

\(<\text{trig\_source}>\):TRig_LeVel2?

**RESPONSE FORMAT**

\(<\text{trig\_source}>\):TRig_LeVel \(<\text{trig\_level}>\)

**EXAMPLE**

The following code adjusts the second trigger level of Channel 3 to 52.00mv:

Command message:  
C3:TRig_LeVel2 52.00mv

**RELATED COMMANDS**

TRIG_COUPLING, TRIG_DELAY,  
TRIG_MODE, TRIG_SELECT, TRIG_SLOPE

Note:  
This command is suitable for non-SPO models.
**DESCRIPTION**

The TRIG_MODE command specifies the trigger mode.

The TRIG_MODE? query returns the current trigger mode.

**Note:**
STOP is a part of the option of this command, but is not a trigger mode of the instrument.

**COMMAND SYNTAX**

TRig_MoDe <mode>

<mode>: = {AUTO, NORM, SINGLE, STOP}

**Note:**
The suffix V is optional and assumed.

**QUERY SYNTAX**

TRig_MoDe?

**RESPONSE FORMAT**

TRig_MoDe <mode>

**EXAMPLE**

The following selects the normal mode:

Command message:
TRMD NORM

**RELATED COMMANDS**

ARM_ACQUISITION, STOP, TRIG_SELECT, TRIG_COUPLING, TRIG_LEVEL, TRIG_SLOP
**TRIGGER**  

**DESCRIPTION**  

The TRIG_SELECT command selects the condition that will trigger the acquisition of waveforms. Depending on the trigger type, additional parameters must be specified. These additional parameters are grouped in pairs. The first in the pair names the variable to be modified, while the second gives the new value to be assigned. Pairs may be given in any order and restricted to those variables to be changed.

The TRIG_SELECT? query returns the current trigger condition.

<table>
<thead>
<tr>
<th>Trigger Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDGE Edge I1 Interval out of range</td>
</tr>
<tr>
<td>GLIT Glitch PL Pulse larger</td>
</tr>
<tr>
<td>INTV Interval PS Pulse smaller</td>
</tr>
<tr>
<td>DROP Dropout P2 Pulse in range</td>
</tr>
<tr>
<td>HT Hold type P1 Pulse out of range</td>
</tr>
<tr>
<td>HV Hold value SR Source</td>
</tr>
<tr>
<td>IL Interval larger TI Time</td>
</tr>
<tr>
<td>IS Interval smaller CHAR Characteristics</td>
</tr>
<tr>
<td>I2 Interval in range LPIC Lines per picture</td>
</tr>
</tbody>
</table>

**COMMAND SYNTAX**  

*(for all but TV trigger)*

TRig_SElect  
<trig_type>,SR,<source>,HT,<hold_type>,HV,<hold_value>  
<trig_type> := { EDGE, GLIT,SLEW, INTV, RUNT, DROP}  
<source> := {C1, C2, C3, C4, LINE,EX,EX5}  
<hold_type> := {TI,PS,PL,P2,P1,IS,IL,I2,I1,OFF}  
<hold_value> := See instrument Operator’s Manual for valid values.

**Note:**  
1. The <hold type> varies with models. If your oscilloscope is an SPO model, hold type’s options are {TI,PS,PL,P2,P1,IS,IL,I2,I1,OFF}, else ,hold type’s options are {TI, PS, PL,PE, IS,
2. When the trigger type is DROP, the hold_type must be TI, or it will make a mistake.

**QUERY SYNTAX**

TRig_SELect?

**RESPONSE FORMAT**

TRig_SELect <trig_type>, SR, <source>, HT, <hold_type>, HV, <hold_value>

**EXAMPLE**

The following selects the EDGE trigger with Channel 1 as trigger source. Hold type and hold-value are chosen as “time” and 1.43US:

Command message:
TRSE EDGE, SR, C1, HT, TI, HV, 1.43US

**TV COMMAND SYNTAX**

Format 1:
TRig_SELect
TV, SR, <source>, CHAR, <standard>, SYNC, <sync_type>, LINE, <line>, FLD, <field>

OPTION:
<trig_type> := {TV}
<source> := {C1, C2, C3, C4}
<standard> :=
{NTSC, PALSEC, 720P/50, 720P/60, 1080P/50, 1080P/60, 1080I/50, 1080I/60, CUSTOM}
<line> := 1 to 525 (PALSEC)
1 to 625 (NTSC)
<br field> := 1 to field_count
<br field_count> := {1, 2, 4, 8}

**Note:**
This format is suitable for SPO models.

Format 2:
TRig_SELect
TV, SR, <source>, CHAR, <standard>, POL, <polarity>, SYNC, <sync_type>, LINE, <line>

OPTION:
<trig_type> := {TV}
<source> := {C1, C2, C3, C4, EX, EX5}
<polarity> := {PO, NE}
PO means positive. NE means negative.
\begin{itemize}
  \item < standard > := \{NTSC, PALSEC\}
  \item <sync_type> := \{AL, LN, OF, EF\}
  \item AL means all lines; LN means line num; OF means odd field; EF means even field.
  \item LINE, <line>: is used to set the line num. if you want to set it. The SYNC must be set to LINENUM
  \item Note:
  \begin{itemize}
    \item This format is suitable for non-SPO models.
  \end{itemize}
\end{itemize}

**TV RESPONSE FORMAT**

\begin{itemize}
  \item TRig\_Select
  \item <trig_type>, SR, <source>, CHAR, <standard>, SY NC, < sync_type >, LINE, <line>, FLD, <field>
\end{itemize}

**RELATED COMMANDS**

\begin{itemize}
  \item TRIG\_COUPLING
  \item TRIG\_DELAY, TRIG\_LEVEL
  \item TRIG\_MODE, TRIG\_SLOPE
\end{itemize}
**TRIGGER**

**TRIG_SLOPE, TRSL**  
Command / Query

**DESCRIPTION**  
The TRIG_SLOPE command sets the trigger slope of the specified trigger source.

The TRIG_SLOPE? query returns the trigger slope of the selected source.

**COMMAND SYNTAX**  
```
<trig_source>: TRig_SLope <trig_slope>
<trig_source>: = {C1, C2, C3, C4, EX,EX5 }
<trig_slope>: = {NEG,POS,WINDOW}
```

**QUERY SYNTAX**  
```
<trig_source>: TRig_Slope?
```

**RESPONSE FORMAT**  
```
<trig_source>: TRig_Slope <trig_slope>
```

**EXAMPLE**  
The following sets the trigger slope of Channel 2 to negative:

Command message:
C2:TRSL NEG

**RELATED COMMANDS**  
TRIG_COUPLING, TRIG_DELAY, TRIG_LEV EL, TRIG_MODE, TRIG_SELECT, TRIG_SLOPE
**DESCRIPTION**

The TRIG_WINDOW command sets the relative height of the two trigger line of the trigger window type.

The TRIG_WINDOW? query returns relative height of the two trigger line of the trigger window type.

**COMMAND SYNTAX**

TRig_WIndow <value>

< value >: -4.5DIV* volt/div to 4.5DIV * volt/div

**QUERY SYNTAX**

TRig_WIndow?

**RESPONSE FORMAT**

TRig_WIndow < value >

**EXAMPLE**

The following sets the relative height of the two trigger line of the trigger window type to 2V:

Command message:

TRWI 2V

**RELATED COMMANDS**

TRIG_LEVEL, TRIG_LEVEL2, TRIG_SE
The TRIG_PATTERN command sets the condition of the pattern trigger.

The TRIG_PATTERN? query returns the condition of the pattern trigger.

**COMMAND SYNTAX**

TRig_Pattern
<source>,<status>[,<source>,<status>][,<source>,<status>][,<source>,<status>],STATE,<condition>

<source> := {C1, C2, C3, C4}

<status> := {X,L,H}

<condition> := { AND, OR, NAND, OR }

**QUERY SYNTAX**

TRig_PAttern?

**RESPONSE FORMAT**

TRig_Pattern<source>,<status>,<source>,<status>,<source>,<status>,STATE,<condition>

**EXAMPLE**

The following sets the channel 2 and channel 3 to low and the condition to AND:

Command message:
TRPA C2,L,C3,L,STATE,AND

**RELATED COMMANDS**

TRIG_LEVEL, TRIG_LEVEL2, TRIG_SELECT

*Note:*
This command is suitable for SPO models.
**CHANNEL**

**UNIT, UNIT**
Command / Query

**DESCRIPTION**

The UNIT command sets the unit of the specified trace.

The UNIT query returns the unit of the specified trace.

**COMMAND SYNTAX**

<channel>:UNIT <type>
<channel>:= {C1, C2, C3, C4}
<type>:= {V,A}

**QUERY SYNTAX**

<channel>:UNIT?

**RESPONSE FORMAT**

<channel>:UNIT <type>

**EXAMPLE**

The following command sets the unit of the channel 1 to V:

Command message:
C1:UNIT V
The VERT_POSITION command adjusts the vertical position of the specified FFT trace on the screen. It does not affect the original offset value obtained at acquisition time.

The VERT_POSITION? query returns the current vertical position of the specified FFT trace.

**COMMAND SYNTAX**

<trace>: Vert_POSITION <display_offset>
<trace>: = {TA, TB, TC, TD}
<display_offset>: =-40 DIV to 40 DIV

Note:
The suffix DIV is optional.

**QUERY SYNTAX**

<trace>: Vert_POSition?

**RESPONSE FORMAT**

<trace>: Vert_POSITION <display_offset>

**EXAMPLE**

The following shifts FFT Trace A (TA) upwards by +3 divisions relative to the position at the time of acquisition:

Command message:
TA: VPOS 3DIV

**RELATED COMMANDS**

FFT_POSITION

Note:
This command is suitable for the non-SPO models.
**CHANNEL VOLT_DIV, VDIV**

**Command/Query**

**DESCRIPTION**
The VOLT_DIV command sets the vertical sensitivity in Volts/div. The VAB bit (bit 2) in the STB register is set if an out-of-range value is entered.

The VOLT_DIV query returns the vertical sensitivity of the specified channel.

**COMMAND SYNTAX**

```
<channel>: Volt_DIV <v_gain>
<channel>:= {C1, C2, C3, C4}
<v_gain>:= 2mV to 10V
```

**Note:**
The suffix V is optional.

**QUERY SYNTAX**

```
<channel>: Volt_DIV?
```

**RESPONSE FORMAT**

```
<channel>: Volt_DIV <v_gain>
```

**EXAMPLE**
The following command sets the vertical sensitivity of channel 1 to 50 mV/div:

Command message:

```
C1: VDIV 50MV
```
WAVEFORM TRANSFER

DESCRIPTION

A WAVEFORM? Query transfers a waveform from the oscilloscope to the controller.

Note:
1. The format of the waveform data depends on the current settings specified by the last WAVEFORM_SETUP command.
2. The format of the waveform data can be seen by the TEMPLATE? Query.

QUERY SYNTAX

<trace>: WaveForm? [<section>]

<trace>:= { C1,C2,C3,C4}

[section] := {DESC, DAT2, ALL}

DESC: Return descriptor. The length of descriptor is 346 bytes. This includes the information necessary to reconstitute the display of the waveform from the data, including: your oscilloscope name and serial number, the encoding format used for the data blocks, and miscellaneous constants.

DAT2: Return the main data include the head, the wave data and the ending flag. The length of data is current memory depth.

ALL: Return the descriptor and data.

RESPONSE FORMAT

<trace>: WaveForm <waveform_data_block>

EXAMPLE

The following command reads waveform data of Channel 1, and current memory depth is 70pts.

Command message:
C1:WF? DAT2
Response message:
As follow picture:
The head of message: C1:WF ALL. These are followed by the string #900000070, the beginning of a binary block in which nine ASCII integers are used to give the length of the block (70 bytes). After the length of block, is beginning of wave data. At the last meet “0A 0A”, means the end of data.

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43 31 3A 57 46 20 41 4C 4C 2C 23 39 30 30 30 30 30 30</td>
<td>C1:WF ALL ,#90000</td>
</tr>
<tr>
<td>30 30 30 37 30 02 03 03 03 03 01 00 FE FC F9</td>
<td>00070..</td>
</tr>
<tr>
<td>F7 F3 F0 ED E9 E6 E3 DF DC D9 D6 D3 D1 CF CE CD</td>
<td>..................</td>
</tr>
<tr>
<td>CC CC CC CD CE CF D1 D4 D6 D9 DC E0 E2 E6 EA ED</td>
<td>..................</td>
</tr>
<tr>
<td>F1 F4 F7 FA FC FE 00 02 02 03 03 03 02 01 00 FE</td>
<td>..................</td>
</tr>
<tr>
<td>FC F9 F6 F3 F0 ED EA E6 E2 DF DC 0A 0A</td>
<td>..................</td>
</tr>
</tbody>
</table>

How to use the data recovery waveform:
1. To calculate the voltage value corresponding to the data point.
   Using the formula: voltage value(V) = code value *( vdiv /25)- voffset.
   code value: The decimal of wave data .
   Note: If the decimal is greater than “127”, it should minus 255. Then the value is code value. Such as the wave data is “FC” convert to decimal is “252”. So the code value is 252-255=-3.
   vdiv: The Volts/div value.
   voffset: The voltage position value.

The following picture as an example:
Send command “C1:VDIV?” ,return” C1:VDIV 5.00E-01V”.
Get the current Volts/div values: vdiv = 0.5V.
Send command “C1:OFST?” ,return” C1:OFST -5.00E-01V”
Get the current voltage position values: voffset = -0.5V.
According the wave data, we can know the first point of wave data is the “02” convert to decimal is “2”(Hexadecimal converted to decimal).
The first point of wave data voltage value = 2*(0.5/25)-(-0.5) = 0.54V.
To calculate the time value of the first data point.

Using the formula: \( \text{time value}(S) = \text{trdl} - (\text{timebase} \times \text{grid}/2) \).

- **trdl**: The time value which is center of the screen.
- **timebase**: The timebase value.
- **grid**: The grid numbers in horizontal direction.

The following picture as an example:

Send command “TRDL?”, return “TRDL -5.000000ns”.

Get the current time value center of the screen: \( \text{trdl} = -5.00 \times 10^{-9} \) s.

Send command “TDIV?”, return “TDIV 5.00E-09S”.

Get the current time base: \( \text{timebase} = 5.00E-09 \) s.

The time value of the first data point: \( \text{time value} = -5.00E-09 - (5.00E-09 \times 14/2) = -40.00E-09 \) s = -40(ns).

Send command “SARA?”, return “SARA 1.00GSa/s”.

Get the current sampling rate: \( \text{sampling rate} = 1.00 \) GSa/s.

The time interval: \( \text{time interval} = 1/ \text{sampling rate} = 1 \) ns

So the time value of the second data point: \( \text{value} = -40 \text{ns} + 1 \text{ns} = -39 \text{ns} \)
RELATED COMMANDS

WAVEFORM_SETUP
**DESCRIPTION**

The **WAVEFORM_SETUP** command specifies the amount of data in a waveform to be transmitted to the controller. The command controls the settings of the parameters listed below.

**Note:**

- **FP** (First point)
- **SP** (Sparsing)
- **NP** (The number of points)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP</td>
<td>First point</td>
</tr>
<tr>
<td>SP</td>
<td>Sparsing</td>
</tr>
<tr>
<td>NP</td>
<td>The number of points</td>
</tr>
</tbody>
</table>

**Sparsing (SP):** The sparsing parameter defines the interval between data points. For example:

- **SP = 0** sends all data points
- **SP = 1** sends all data points
- **SP = 4** sends every 4th data point

**Number of points (NP):** The number of points parameter indicates how many points should be transmitted. For example:

- **NP = 0** sends all data points
- **NP = 1** sends 1 data point
- **NP = 50** sends a maximum of 50 data points
- **NP = 1001** sends a maximum of 1001 data points

**First point (FP):** The first point parameter specifies the address of the first data point to be sent. For waveforms acquired in sequence mode, this refers to the relative address in the given segment. For example:

- **FP = 0** corresponds to the first data point
- **FP = 1** corresponds to the second data point
- **FP = 5000** corresponds to data point 5001

The **WAVEFORM_SETUP?** query returns the transfer parameters currently in use.

**COMMAND SYNTAX**

Usage1:

```
WaveForm_SetUp
SP,<sparsing>,NP,<number>, FP, <point>
```
Usage2:
WaveForm_SetUp TYPE,<len>

<len> : = {0, 1}

Note:
1. For SPO models, you can use the usage2 to control the returned waveform data, 0 means all waveform data of screen, 1 means all waveform data of memory depth.

QUERY SYNTAX

WaveForm_SetUp?

Note:
1. Parameters are grouped in pairs. The first of the pair names the variable to be modified, whilst the second gives the new value to be assigned. Pairs may be given in any order and may be restricted to those variables to be changed.
2. After power-on, SP is set to 4, NP is set to 100, and FP is set to 0.

RESPONSE FORMAT

WaveForm_SetUp
SP,<sparsing>,NP,<number>,FP,<point>

EXAMPLE

The following command specifies that every 3rd data point (SP=3) starting at address 200 should be transferred:
Command message:
WFSU SP, 3, FP, 200

RELATED COMMANDS

WAVEFORM
The XY_DISPLAY command enables or disables the display the XY format.

The response to the XY_DISPLAY? query indicates whether the XY format display is enabled.

**COMMAND SYNTAX**

```
XY_DISPLAY <state>
<state> := {ON, OFF}
```

**QUERY SYNTAX**

```
XY_DISPLAY?
```

**RESPONSE FORMAT**

```
XY_DISPLAY <state>
```

**EXAMPLE**

The following command enables to display the XY format:

Command message:

XYDS ON
Programming Examples

This chapter gives some examples for the programmer. In these examples you can see how to use the NI-VISA lib or sockets and the commands which have been described before this chapter to control our devices. By the examples’ guide, you can develop more functions application as you want. This example is developed by Visual Studio project.

- Example of VC++
- Example of VB
- Example of MATLAB
- Example of LabVIEW
- Example of C#
Example of VC++

Environment: Win7 32bit system, Visual Studio

The functions of this example: use the NI-VISA, to control the device with USBTMC or TCP/IP access to do a write and read.

Follow the steps to finish the example:

1. Open Visual Studio, create a new VC++ win32 project.
2. Set the project environment to use the NI-VISA lib, there are two ways to use NI-VISA, static or automatic:
   2.1 Static: find files: visa.h, visatype.h, visa32.lib in NI-VISA install path. Copy them to your project, and add them into project. In the projectname.cpp file, add the follow two lines:
   ```cpp
   #include "visa.h"
   #pragma comment(lib,"visa32.lib")
   ```
   2.2 Automatic:
   Set the .h file include directory, the NI-VISA install path, in our computer we set the path is : C:\Program Files\IVI Foundation \VISA\WinNT\include. Set this path to project---properties---c/c++---General---Additional Include Directories: See the picture.
Set lib path set lib file:
Set lib path: the NI-VISA install path, in our computer we set the path is: C:\Program Files\IVI Foundation\VISA\WinNT\lib\msc. Set this path to project---properties---Linker---General---Additional Library Directories: as seen in the pictures below.

Set lib file: project---properties---Linker---Command Line---Additional Options: visa32.lib
Include visa.h file: In the projectname.cpp file:

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

3. Add codes:

3.1 USBTMC access code:

Write a function Usbtmc_test:

```cpp
Int Usbtmc_test()
{
    /* This code demonstrates sending synchronous read & write commands */
    /* to an USB Test & Measurement Class (USBTMC) instrument using */
    /* NI-VISA */
    /* The example writes the "*IDN?\n" string to all the USBTMC */
    /* devices connected to the system and attempts to read back */
```
/* results using the write and read functions. */
/* The general flow of the code is */
/* Open Resource Manager */
/* Open VISA Session to an Instrument */
/* Write the Identification Query Using viPrintf */
/* Try to Read a Response With viScanf */
/* Close the VISA Session */
/**************************************************************************/

ViSessiondefaultRM;
ViSessioninstr;
ViUInt32numInstrs;
ViFindListfindList;
ViUInt32retCount;
ViUInt32writeCount;
ViStatusstatus;
CharinstrResourceString[VI_FIND_BUFLEN];
Unsignedcharbuffer[100];
Charstringinput[512];
Inti;
/** First we must call viOpenDefaultRM to get the manager */
* handle. We will store this handle in defaultRM.*/
status=viOpenDefaultRM (&defaultRM);
if (status<VI_SUCCESS)
{
    printf ("Could not open a session to the VISA Resource
Manager!\n");
    return status;
}

/* Find all the USB TMC VISA resources in our system and store the
number of resources in the system in numInstrs. */
status = viFindRsrc (defaultRM, "USB?*INSTR", &findList,
                        &numIntrs, instrResourceString);
if (status<VI_SUCCESS)
{
    printf ("An error occurred while finding resources.\nHit enter to

continue.\n");
    fflush(stdin);
    getchar();
    viClose (defaultRM);
    return status;
}

/** Now we will open VISA sessions to all USB TMC instruments.
* We must use the handle from viOpenDefaultRM and we must
* also use a string that indicates which instrument to open.  This
* is called the instrument descriptor.  The format for this string
Digital Oscilloscopes Series

* can be found in the function panel by right clicking on the
* descriptor parameter. After opening a session to the
* device, we will get a handle to the instrument which we
* will use in later VISA functions. The AccessMode and Timeout
* parameters in this function are reserved for future
* functionality. These two parameters are given the value
VI_NULL.*/

```c
for (i=0; i<numInstrs; i++)
{
    if (i> 0)
        viFindNext (findList, instrResourceString);
    status = viOpen (defaultRM, instrResourceString, VI_NULL,
                   VI_NULL, &instr);
    if (status<VI_SUCCESS)
    {
        printf ("Cannot open a session to the device %d.
", i+1);
        continue;
    }
/* * At this point we now have a session open to the USB TMC
instrument.
* We will now use the viPrintf function to send the device the string
"*IDN?\n",}
* asking for the device's identification. */

char * command ="*IDN?\n";

status = viPrintf (instr, command);

if (status<VI_SUCCESS)
{
    printf ("Error writing to the device %d.\n", i+1);
    status = viClose (instr);
    continue;
}

/** Now we will attempt to read back a response from the device to
* the identification query that was sent. We will use the viScanf
* function to acquire the data.
* After the data has been read the response is displayed.*/

status = viScanf(instr, "%t", buffer);

if (status<VI_SUCCESS)
    printf ("Error reading a response from the device %d.\n", i+1);
else
    printf ("\nDevice %d: %*s\n", i+1,retCount, buffer);

status = viClose (instr);

/** Now we will close the session to the instrument using
* viClose. This operation frees all system resources. */
status = viClose (defaultRM);
    return 0;
}

3.2 TCP/IP access code:
    Write a function TCP_IP_Test.

Int TCP_IP_Test(char *pIP)
{
    Char outputBuffer[VI_FIND_BUFLEN];
    ViSession defaultRM, instr;
    ViStatus status;
    ViUInt32 count;
    ViUInt16 portNo;

    /* First we will need to open the default resource manager. */
    status = viOpenDefaultRM (&defaultRM);
    if (status<VI_SUCCESS)
    {
        printf("Could not open a session to the VISA Resource Manager!\n");
    }

    /* Now we will open a session via TCP/IP device */
    Char head[256] ="TCPIP0::";
    Chartail[] =":INSTR";
Char resource [256];
strcat(head,pIP);
strcat(head,tail);
status = viOpen (defaultRM, head, VI_LOAD_CONFIG, VI_NULL, &instr);
if (status<VI_SUCCESS)
{
    printf("An error occurred opening the session\n");
    viClose(defaultRM);
}
status = viPrintf(instr, "*idn?\n");
status = viScanf(instr, "%t", outputBuffer);
if (status<VI_SUCCESS)
{
    printf("viRead failed with error code: %x \n",status);
    viClose(defaultRM);
} else
    printf("ndata read from device: %*s\n", 0,outputBuffer);
status = viClose (instr);
status = viClose (defaultRM);
return 0;
}
Example of VB

Environment: Win7 32bit system, Microsoft Visual Basic 6.0

The function of this example: Use the NI-VISA, to control the device with USBTMC and TCP/IP access to do a write and read.

Follow the steps to complete the example:

1. Open Visual Basic, build a standard application program project (Standard EXE)
2. Set the project environment to use the NI-VISA lib, Click the Existing tab of Project>>Add Module. Search for the visa32.bas file in the include folder under the NI-VISA installation path and add the file.

This allows the VISA functions and VISA data types to be used in a program.
3. Add codes:

3.1. USBTMC access code:

Write a function Usbtmc_test.

Private Function Usbtmc_test() As Long

' This code demonstrates sending synchronous read & write
commands
' to an USB Test & Measurement Class (USBTMC) instrument
using
' NI-VISA
' The example writes the "*IDN?\n" string to all the USBTMC
' devices connected to the system and attempts to read back
' results using the write and read functions.
' The general flow of the code is
' Open Resource Manager
' Open VISA Session to an Instrument
' Write the Identification Query Using viWrite
' Try to Read a Response With viRead
' Close the VISA Session

Const MAX_CNT = 200

Dim defaultRM As Long

Dim instrsesn As Long
Dim numInstrs As Long
Dim findList As Long
Dim retCount As Long
Dim writeCount As Long
Dim status As Long
Dim instrResourceString As String * VI_FIND_BUFLEN
Dim buffer As String * MAX_CNT
Dim i As Integer

' First we must call viOpenDefaultRM to get the manager handle. We will store this handle in defaultRM.
status = viOpenDefaultRM(defaultRM)
If (status < VI_SUCCESS) Then
    Debug.Print "Could not open a session to the VISA Resource Manager!"
    Usbtmc_test = status
    ExitFunction
End If

' Find all the USB TMC VISA resources in our system and store the
' number of resources in the system in numInstrs.
status = viFindRsrc(defaultRM, "USB?*INSTR", findList, numIntrs, instrResourceString)

If (status < VI_SUCCESS) Then
    Debug.Print "An error occurred while finding resources."
    viClose (defaultRM)
    Usbtmc_test = status

Exit Function

End If

' Now we will open VISA sessions to all USB TMC instruments.
' We must use the handle from viOpenDefaultRM and we must
' also use a string that indicates which instrument to open. This
' is called the instrument descriptor. The format for this string
' can be found in the function panel by right clicking on the
' descriptor parameter. After opening a session to the
' device, we will get a handle to the instrument which we
' will use in later VISA functions. The AccessMode and Timeout
' parameters in this function are reserved for future
' functionality. These two parameters are given the value

VI_NULL.

For i = 0 To numInstrs
    If (i > 0) Then
status = viFindNext(findList, instrResourceString)

End If

status = viOpen(defaultRM, instrResourceString, VI_NULL, VI_NULL, instrsesn)
If (status < VI_SUCCESS) Then
    Debug.Print "Cannot open a session to the device ", i + 1
    GoTo NextFind
End If

' At this point we now have a session open to the USB TMC instrument.
' We will now use the viWrite function to send the device the string "*IDN?",
' asking for the device's identification.
status = viWrite(instrsesn, "*IDN?", 5, retCount)
If (status < VI_SUCCESS) Then
    Debug.Print "Error writing to the device."
    status = viClose(instrsesn)
    GoTo NextFind
End If
' Now we will attempt to read back a response from the device to
'the identification query that was sent. We will use the viRead
' function to acquire the data.
' After the data has been read the response is displayed.
status = viRead(instrsesn, buffer, MAX_CNT, retCount)
If (status < VI_SUCCESS) Then
    Debug.Print "Error reading a response from the device.", i + 1
Else
    Debug.Print i + 1, retCount, buffer
End If
    status = viClose(instrsesn)
NextFind:
Next i

' Now we will close the session to the instrument using
' viClose. This operation frees all system resources.
status = viClose(defaultRM)
Usbtmc_test = 0
End Function
3.2 TCP/IP access code:

Write a function TCP_IP_Test.

**Private Function** TCP_IP_Test(ip As String) As Long

**Dim** outputBuffer As String * VI_FIND_BUFLEN

**Dim** defaultRM As Long

**Dim** instrsesn As Long

**Dim** status As Long

**Dim** count As Long

' First we will need to open the default resource manager.

    status = viOpenDefaultRM (defaultRM)

    If (status < VI_SUCCESS) Then

        Debug.Print "Could not open a session to the VISA Resource Manager!"

        TCP_IP_Test = status

        Exit Function

    End If

' Now we will open a session via TCP/IP device

    status = viOpen(defaultRM, "TCPIP0::" + ip + "::INSTR",
                   VI_LOAD_CONFIG, VI_NULL, instrsesn)

    If (status < VI_SUCCESS) Then
Debug.Print "An error occurred opening the session"

viClose (defaultRM)

TCP_IP_Test = status

Exit Function

End If

status = viWrite(instrsesn, "*IDN?", 5, count)

If (status < VI_SUCCESS) Then
    Debug.Print "Error writing to the device."
End If

status = viRead(instrsesn, outputBuffer, VI_FIND_BUFLEN, count)

If (status < VI_SUCCESS) Then
    Debug.Print "Error reading a response from the device.", i + 1
Else
    Debug.Print "read from device:", outputBuffer
End If

status = viClose(instrsesn)

status = viClose(defaultRM)

TCP_IP_Test = 0

End Function
Example of MATLAB

Environment: Win7 32bit system, MATLAB R2010b

The function of this example: Use the NI-VISA, to control the device with USBTMC or TCP/IP access to do a write and read.

Follow the steps to complete the example:

Open MATLAB, modify the current directory. In this demo, the current directory is modified to D:\USBTMC_TCPIP_Demo.

Click File>>New>>Script in the Matlab interface to create an empty M file

Add codes:

USBTMC access code:

Write a function Usbtmc_test.

```matlab
function USBTMC_test()
    % This code demonstrates sending synchronous read & write commands
    % to an USB Test & Measurement Class (USBTMC) instrument using % NI-VISA

    %Create a VISA-USB object connected to a USB instrument
    vu = visa('ni','USB0::0xF4EC::0xEE38::0123456789::INSTR');

    %Open the VISA object created
    fopen(vu);
```
Send the string "*IDN?", asking for the device's identification.
fprintf(vu, '*IDN?');

Request the data
outputbuffer = fscanf(vu);
disp(outputbuffer);

Close the VISA object
fclose(vu);
delete(vu);
clear vu;
end

3.2 TCP/IP access code:
Write a function TCP_IP_Test.

function TCP_IP_test(IPstr)
% This code demonstrates sending synchronous read & write commands
% to an TCP/IP instrument using NI-VISA

Create a VISA-TCPIP object connected to an instrument
% configured with IP address.
vt = visa('ni', ['TCPIP0::', IPstr, '::INSTR']);

Open the VISA object created
fopen(vt);

Send the string "*IDN?", asking for the device's identification.
fprintf(vt, '*IDN?');
% Request the data
outputbuffer = fscanf(vt);
disp(outputbuffer);

% Close the VISA object
fclose(vt);
delete(vt);
clear vt;

dend
Example of LabVIEW

Environment: Win7 32bit system, LabVIEW 2011

The functions of this example: use the NI-VISA, to control the device with USBTMC and TCP/IP access to do a write and read.

Follow the steps to complete the example:

1. Open LabVIEW, create a VI file.
2. Add controls. Right-click in the Front Panel interface, select and add VISA resource name, error in, error out and some indicators from the Controls column.
3. Open the Block Diagram interface. Right-click on the VISA resource name and you can select and add the following functions from VISA Palette from the pop-up menu: VISA Write, VISA Read, VISA Open and VISA Close.
4. Connect them as shown in the figure below

5. Select the device resource from the VISA Resource Name
list box and run the program.

In this example, the VI opens a VISA session to a USBTMC device, writes a command to the device, and reads back the response. In this example, the specific command being sent is the device ID query. Check with your device manufacturer for the device command set. After all communication is complete, the VI closes the VISA session.

6. Communicating with the device via TCP/IP is similar to USBTMC. But you need to change VISA Write and VISA Read Function to Synchronous I/O. The LabVIEW default is asynchronous I/O. Right-click the node and select Synchronous I/O Mod >> Synchronous from the shortcut menu to write or read data synchronously.

7. Connect them as shown in the figure below
8. Input the IP address and run the program.
Example of C#

Environment: Win7 32bit system, Visual Studio

The functions of this example: use the NI-VISA, to control the device with USBTMC or TCP/IP access to do a write and read.

Follow the steps to finish the example:

1、Open Visual Studio, create a new C# project.
2、Add References. Add NationalInstruments.Common.dll and NationalInstruments.VisaNS.dll to the project. (Notice: you must install the .NET Framework 3.5/4.0/4.5 Languages support when you install the NI-VISA.)

3、Write C# Code

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```
using National Instruments.VisaNS;

namespace TestVisa
{
    class Program
    {
        static void Main(string[] args)
        {
            // Find all the USBTMC resources
            string[] usbRsrcStrings = ResourceManager.GetLocalManager().FindResources("USB?*INSTR");
            if (usbRsrcStrings.Length <= 0)
            {
                Console.WriteLine("Can not find USBTMC Device!");
                return;
            }

            // Choose the first resource string to connect the device.
            // You can input the address manually
            // USB TMC:
            // MessageBasedSession mbSession =
            // (MessageBasedSession)ResourceManager.GetLocalManager().Open("USB 0::0xF4EC::0xEE38::0123456789::INSTR");
            // TCP IP:
            // MessageBasedSession mbSession =
            // (MessageBasedSession)ResourceManager.GetLocalManager().Open("TCP IP0::192.168.1.100::INSTR");
            MessageBasedSession mbSession = (MessageBasedSession)ResourceManager.GetLocalManager().Open(usbRsrcStrings[0]);
            mbSession.Write("*IDN?");
            string result = mbSession.ReadString();
            mbSession.Dispose();
        }
    }
}
```csharp
Console.WriteLine(result);
```
Using Sockets Examples

As mentioned above, socket communication is supported by the operating system—and it is straightforward. Note that SCPI strings are terminated with a “\n” (new line) character.

**Example of C**

```c
int MySocket;
if((MySocket=socket(PF_INET,SOCK_STREAM,0))==-1) exit(1);
struct in_addr {
    unsigned long s_addr;
};
struct sockaddr_in {
    short int sin_family; // Address family
    unsigned short int sin_port; // Port number
    struct in_addr sin_addr; // Internet address
    unsigned char sin_zero[8]; // Padding
};
struct sockaddr_in MyAddress;
// Initialize the whole structure to zero
memset(&MyAddress,0,sizeof(struct sockaddr_in));
// Then set the individual fields
MyAddress.sin_family=PF_INET; // IPv4
MyAddress.sin_port=htons(5025); // Port number used by most instruments
MyAddress.sin_addr.s_addr=inet_addr("169.254.9.80"); // IP Address

// Establish TCP connection
if(connect(MySocket,(struct sockaddr *)&MyAddress,sizeof(struct sockaddr_in))==-1) exit(1);
```
// Send SCPI command
if(send(MySocket,"*IDN?\n",6,0)==-1) exit(1);

// Read response
char buffer[200];
int actual;
if((actual=recv(MySocket,&buffer[0],200,0))== -1) exit(1);
buffer[actual]=0; // Add zero character (C string)
printf("Instrument ID: %s\n",buffer);

// Close socket
if(close(MySocket)==-1) exit(1);
Index

A
ACQUIRE_WAY, ACQW, Command/Query,
ALL_STATUS?, ALST?, Query,
ARM_ACQUISITION, ARM, Command,
ATTENUATION, ATTN, Command/Query,
AUTO_CALIBRATE, ACAL, Command/Query,
AUTO_SETUP, ASET, Command,
AUTO_TYPESET, AUTTS, Command/Query,
AVERAGE_ACQUIRE, AVGA, Command/Query,

B
BANDWIDTH_LIMIT, BWL, Command/Query,
BUZZER, BUZZ, Command,

C
CAL?, Query,
COMM_HEADER, CHDR, Command/Query,
CLS, Command,
CMR?, Query,
COMM_NET, CONET, Command/Query,
COUPLING, CPL, Command/Query,
CURSOR_MEASURE, CRMS, Command/Query,
CURSOR_SET, CRST, Command/Query,
CURSOR_TYPE, CRTY, Command/Query,
CURSOR_VALUE?, CRVA?, Query,
CSV_SAVE, CSVS, Command/Query,
CYMOMETER, CYMT, Query,

D
DATE, Command/Query,
DDR?, Query,
DEFINE, DEF, Command/Query,
DOT_JOIN, DTJN, Command/Query,

E
ESE, Command/Query,
ESR?, Query,
EXR?, Query,

F
FRAME_PARAM?, FPAR?, Query,
FRAME_SET, FRAM, Command,
FRAME_TIME?, FTIM?, Query,
FILTER, FILT, Command/Query,
FILT_SET, FILTS, Command/Query,
FFT_WINDOW, FFTW, Command/Query,
FFT_ZOOM, FFTZ, Command/Query,
FFT_TDIV, FFTT, Command/Query,
FFT_UNIT, FFTU, Command/Query,
FFT_CENTER, FFTC, Command/Query,
FFT_SCALE, FFTS, Command/Query,
FFT_FULLSCREEN, FFTF, Command/Query,
FFT_POSITION, FFTP, Command/Query,

G
GRID_DISPLAY, GRDS, Command/Query,

H
HOR_MAGNIFY, HMAG, Command/Query,
HOR_POSITION, HPOS, Command/Query,

I
IDN?, Query,
INTENSITY, INTS, Command/Query,
INR, INR, Query,
INVERT_SET, INVS, Command/Query,

M
MENU, MENU, Command/Query,
MATH_VERT_POS, MTVP, Command/Query,
MATH_VERT_DIV, MTVD, Command/Query,
MEASURE_DELY, MEAD, Command/Query,
MEMORY_SIZE, MSIZ, Command/Query,

O
OFFSET, OFST, Command/Query,
OPC, Command/Query,

P
PARAMETER_CLR, PACL, Command,
PARAMETER_CUSTOM, PACU, Command/Query,
PARAMETER_VALUE?, PAVA?, Query,
PERSIST_SETUP, PESU, Command/Query,
PANEL_SETUP, PNSU, Command/Query,
PF_DISPLAY, PFDS, Command/Query,
PF_SET, PFST, Command/Query,
PF_CONTROL, PFCT, Command/Query,
PF_SOURCE, PFSC, Command/Query,
PF_BUFFER, PFBF, Command/Query,
PF_OPERATION, PFOP, Command/Query,
PF_FAIL_STOP, PFFS, Command/Query,
PF_CREATEM, PFCM, Command,
PF_DATEDIS, PFDD, Query,

R
RCL, Command,
RECALL, REC, Command,
RECALL_PANEL, RCPN, Command,
RST, Command,
REF_SET, REFS, Command/Query,
REF_CLOSE, REFCL, Command,
REF_DISPALY, REFDI, Command/Query,
REF_LOCATION, REFLA, Command/Query,
REF_POISITION, REFP0, Command/Query,
REF_SAVE, REFSA, Command,
REF_SCLALE, REFSC, Command/Query,
REF_SOURCE, REFSR, Command/Query,

S
SAV, Command,
SCREEN_DUMP, SCDP, Command/Query,
SCREEN_SAVE, SCSV, Command/Query,
SRE, Command/Query,
STB?, Query,
STOP, Command,
STORE_PANEL, STPN, Command,
SAMPLE_STATUS, SAST/ Query,
SAMPLE_RATE, SARA/ Query,
SAMPLE_NUM, SANU/ Query,
SKEW, SKEW, Command,
SET%50, SET50, Command,
SINXX_SAMPLE, SXSA, Command/Query,

T
TIME_DIV, TDIV, Command/Query,
TEMPLATE?, TMPL?, Query,
TRACE, TRA, Command/Query,
TRIG_COUPLING, TRCP, Command/Query,
TRIG_DELAY, TRDL, Command/Query,
TRIG_LEVEL, TRLV, Command/Query,
TRIG_LEVEL2, TRLV2, Command/Query,
TRIG_MODE, TRMD, Command/Query,
TRIG_SELECT, TRSE, Command/Query,
TRIG_SLOPE, TRSL, Command/Query,
TRIG_WINDOW, TRWI, Command/Query,
TRIG_PATTERN, TRPA, Command/Query,

U
UNIT, UNIT, Command/Query,

V
VERT_POSITION, VPOS, Command/Query,
VOLT_DIV, VDIV, Command/Query,

W
WAVEFORM, WF, Command/Query,
WAVEFORM_SETUP, WFSU, Command/Query,

X
XY_DISPLAY, XYDS, Command/Query,