Acute LAVISA

Version 1.0.9 rev. 3

Acute Technology Inc.

Copyright © 2025 Acute Technology Inc.

Table of contents

1. Introduction 4
1.1 Scope 4
1.2 Core Concepts 5
1.3 Supported Platforms 6
1.3.1 Windows 6
1.3.2 Linux 6
1.4 Interfaces 7
2. Basic Control 8
2.1 Flow Chart 8
2.2 Installation 9
2.2.1 Overview 9
2.2.2 Installation Steps Instructions 9
2.3 DLL Functions 10
2.3.1 viSelectAppType 10
2.3.2 viOpenRM 10
2.3.3 viWrite 10
2.3.4 viRead 11
2.3.5 viGetCommandResult 11
2.3.6 viCloseRM 11
2.3.7 viErrCode 11
3. Remote Control 12
3.1 Overview 12
3.2 Quick Start 13
3.2.1 Launching gRPC Server 13
3.2.2 Python Sample Client 14
3.3 Protobuf Definition 16
3.3.1 Remote Procedure Calls (RPCs) 16
3.3.2 Messages 16
3.3.3 Enums 17
4. Supported Commands 18
4.1 Common Commands 18
4.1.1 Command Formats 18
4.1.2 Command Definitions 19
4.2 File Subsystem 21
4.2.1 Command Formats 21

Copyright © 2025 Acute Technology Inc.

4.2.2 Command Definitions 21
4.3 Window Subsystem 28
4.3.1 Command Formats 28
4.3.2 Command Definitions 28
4.4 Logic Analyzer Mode Subsystem 30
4.4.1 What is Logic Analyzer Mode? 30
4.4.2 Command Formats 30
4.4.3 Command Definitions 33
4.5 Protocol Analyzer Mode Subsystem 71
4.5.1 What is Protocol Analyzer Mode? 71
4.5.2 Command Formats 71
4.5.3 Command Definitions 72
5. Error Codes 78
5.1 Status Code Table 78
6. Release Notes 81

Copyright © 2025 Acute Technology Inc.

1. Introduction

1.1 Scope

This is the full documentation for Acute LAVISA (simplified as AqLAVISA).

The following topics are in scope for this documentation:

- AqLAVISA basic concepts and features
- Interfaces on top of AqLAVISA and their usage
- Supported commands and their usage

The following topics are out of scope for this documentation:

- System design details of third party applications
- Protocol specific decoding and analysis details

AqLAVISA offers an intuitive interface for seamless management of controlling Acute's logic analyzers. Developers can utilize this interface to capture, analyze, and save the captured data from logic analyzers.

Copyright © 2025 Acute Technology Inc.

1.2 Core Concepts

AqLAVISA provides API interfaces for user program to communicate with the Acute Logic Analyzer software by using text commands, user could access the protocol analysis report in Protocol Analyzer Page, decode report or Timing Sequence Analyzer report from Logic Analyzer Page by using this interface. For automatic testing, developers are able to setup the capture and environment settings by load a sample waveform file, then start capture to record new data, and read the decode or analysis information to check and verify the target device’s behavior.

AqLAVISA interface allows developers to automate software by sending pure text commands. Developers are able to collect these data such as decode information, timing measurements after acquisition process. Originally, this interface is exposed via shared library (.dll) that handles all received commands and communicating with the software. However, a problem arises that it is limited locally to a single machine which the instrument is physically connected to. Therefore, remote controlling is a way to make it more versatile and flexible for users.

Copyright © 2025 Acute Technology Inc.

1.3 Supported Platforms

1.3.1 Windows

Software	Version
MSO	1.7.82 or above
MSO (Full)	1.7.82 or above
BusFinder and Logic Analyzer	1.7.74 or above
TravelLogic	1.7.47 or above
TravelBus	1.7.34 or above

1.3.2 Linux

> **Note**
>
> The application on Linux platform are still marked as beta versions. Please create an issue if you encounter any errors when running the program.

Software	Version
Mixed_Signal_Oscilloscope	2.0.0 or above
Digital_Storage_Oscilloscope	2.0.0 or above
BusFinder_Logic_Analyzer	2.0.0 or above
TravelLogic_Analyzer	2.0.0 or above
TravelBus_Analyzer	2.0.0 or above

Copyright © 2025 Acute Technology Inc.

1.4 Interfaces

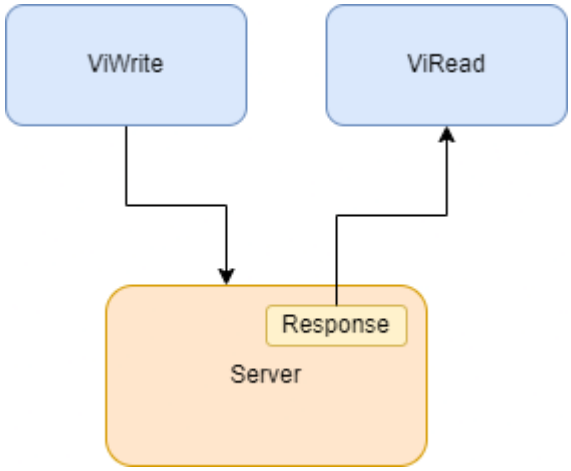
AqLAVISA provides three different interfaces to interact with the logic analyzer. Each interface has links to the following instructions, tutorials, and API references. The following interfaces are supported:

- Local Socket Inter-process Communication: Windows platforms only
- TCP Socket Connection: Windows platforms only
- gRPC

Copyright © 2025 Acute Technology Inc.

2. Basic Control

2.1 Flow Chart



Simply send commands via viWrite call. The server will generate a response after the command is fully executed. If there is queried value to be returned, you can retrieve the value by using viRead call.

Copyright © 2025 Acute Technology Inc.

2.2 Installation

2.2.1 Overview

AqVISA consists of several parts:

- A dynamic library (DLL) file

  `AqVISA64.dll` for 64-bit Windows systems

- Library dependency files

Component	Filename
Qt 5 Core Library	`Qt5Core.dll`
Qt 5 Network Library	`Qt5Network.dll`
ICU Libraries	`icudt.dll`, `icuin.dll`, `icuuc.dll`
C++ Runtime Library	`msvcp120.dll`, `msvcr120.dll`

- User Manual

  `AqVISA_UserManual_en.pdf`

- A Python Package for Python users

  PyPI Package: aqvisa

  GitHub Repository: aqvisa-python

2.2.2 Installation Steps Instructions

- Run the installer.

  Please refer to the installation page to download the installer.

  By default, the installer will install the dynamic library and the dependency libraries to the following directory:

  ```
  C:\Users\<username>\AppData\Local\Acute\SDK\AqVISA
  ```

  The installer has an option to add the installation directory to the system PATH. This is recommended for most users to locate the dynamic library easily.

- (Optional) Install the Python Package

  You can install aqvisa as a python package using a package manager called `pip`, which installs packages from the Python Packaging Index (PyPI).

  ```
  pip install aqvisa
  ```

- Sample Usage

  Please refer to the sample code in the GitHub repository.

  Python

Copyright © 2025 Acute Technology Inc.

2.3 DLL Functions

2.3.1 viSelectAppType

```
int viSelectAppType(int iAppType)
```

Setup the target connection Acute Logic Analyzer software.

Parameters

- `iAppType [In]`:

  Type: ***int***

  Available parameters are listed below

Parameter	Target software	Recommended S/W Ver.
APP_TYPE_TL	0	TL (x64).exe / TL.exe
APP_TYPE_BF	1	BF_LA3K(x64).exe / BF_LA3K.exe
APP_TYPE_TB	2	TBA (x64).exe / TBA.exe
APP_TYPE_MS	3	MSO (x64).exe / MSO.exe

Return value

Return 1 if success; otherwise, return 0.

2.3.2 viOpenRM

```
int viOpenRM()
```

Connect to the Logic Analyzer software.

Return value

Return 1 if success; otherwise, return 0.

2.3.3 viWrite

```
int viWrite( char* szCommand )
```

Send AqLAVISA command to the Logic Analyzer software.

Parameters

- `szCommand [In]`:

  Type: ***char ****

  Contains AqLAVISA command, for more detail, please refer to Acute LA VISA Command List.

Return value

Return 1 if success; otherwise, return 0.

Copyright © 2025 Acute Technology Inc.

2.3.4 viRead

```
int viRead(char *szReturnBuffer, int iBufferLength )
```

Read AqLAVISA command execute result from the Logic Analyzer software.

Parameters

- `szReturnBuffer [Out]`:

  Type: ***char ****

  Data buffer for the AqLAVISA command returning value.

- `iBufferLength [In]`:

  Type: ***int***

  Buffer size of szReturnBuffer.

Return value

Return the number of bytes read.

2.3.5 viGetCommandResult

```
int viGetCommandResult()
```

Read the command processing result of previous AqLAVISA viWrite function call.

Return value

Return	value	Description
PROC_OK_WOUT_RETURN	0	Process successed with no returned data.
PROC_OK_WITH_RETURN	1	Process successed with returned data.
PROC_FAIL	2	Process failed.
PROC_PENDING	3	In process.

2.3.6 viCloseRM

```
int viCloseRM()
```

Disable the AqLAVISA function and disconnect from the Logic Analyzer software.

Return value

Return 1 if success; otherwise, return 0.

2.3.7 viErrCode

```
int viErrCode()
```

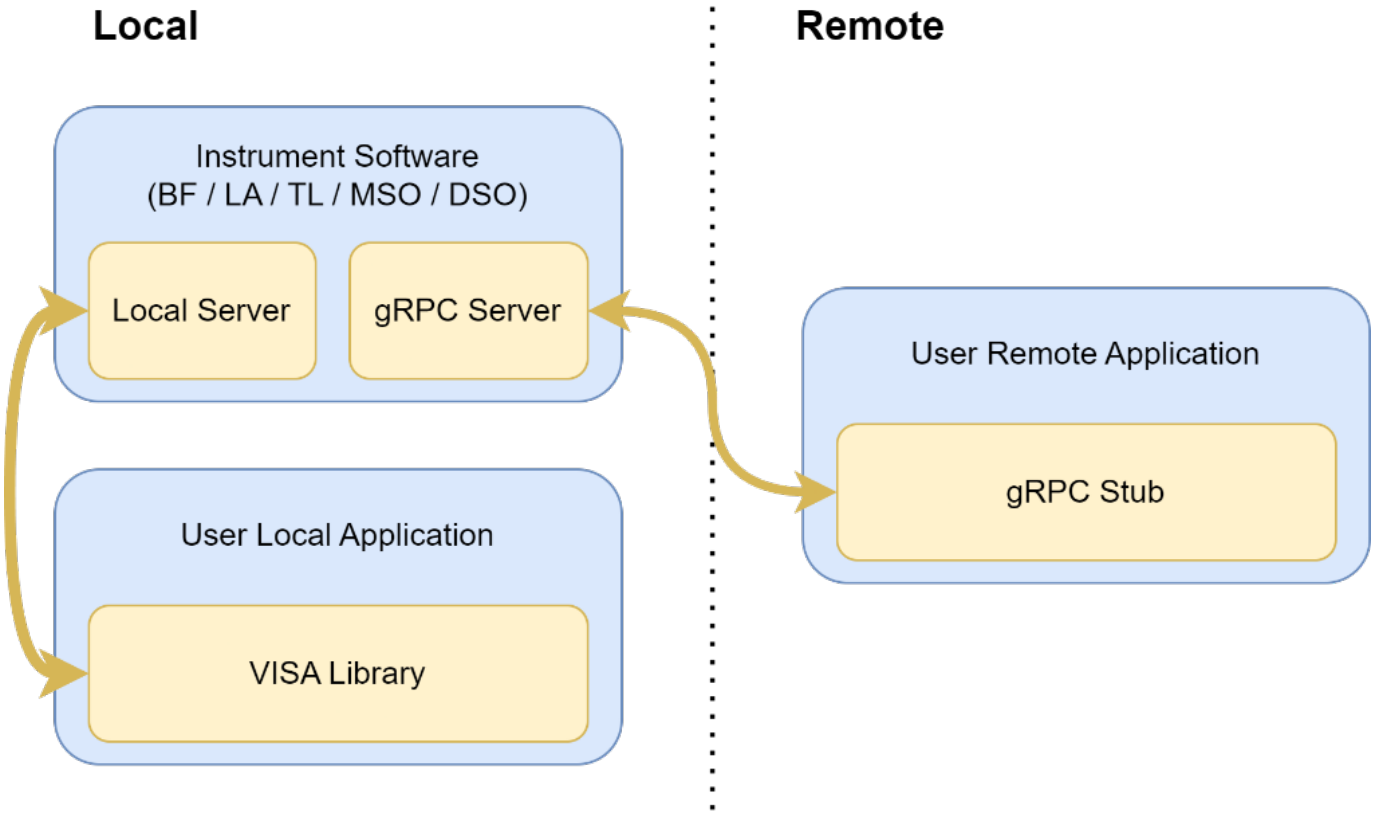
Retrieve the AqLAVISA error code. Please refer to the Error Code for more details.

Copyright © 2025 Acute Technology Inc.

3. Remote Control

3.1 Overview

As in many automation system, AqVISA keeps the interface always consistent and augmenting its feature by allowing new types of text commands. It is overall around the idea of client-server model. On the server side, the instrument software implements the AqVISA interface and runs the server to handle incoming commands. Here we can see in the block diagram below that the software runs local server and gRPC server to handle requests either from local application or remote application.



On the client side, developers can build application with any language they prefer, since multi-language is also basic feature supported by gRPC. So, for example, you can easily create a client application in either Python, C#, etc.

In this section, we will only cover the setup regarding remote controlling via gRPC.

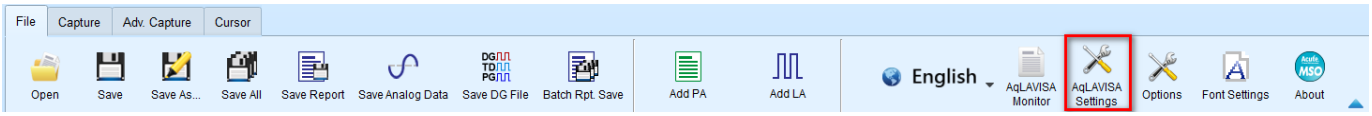
Copyright © 2025 Acute Technology Inc.

3.2 Quick Start

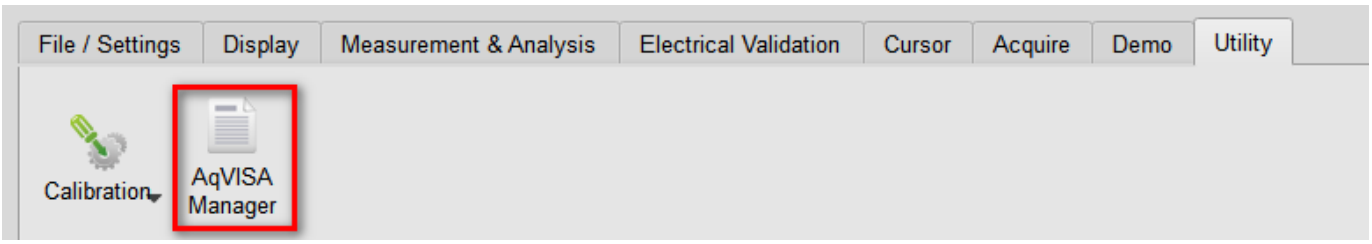
3.2.1 Launching gRPC Server

gRPC server can be enabled either in the software UI or by command line

For BusFinder / Logic Analyzer / TravelLogic / MSO software, select **File > AqLAVISA Manager**.

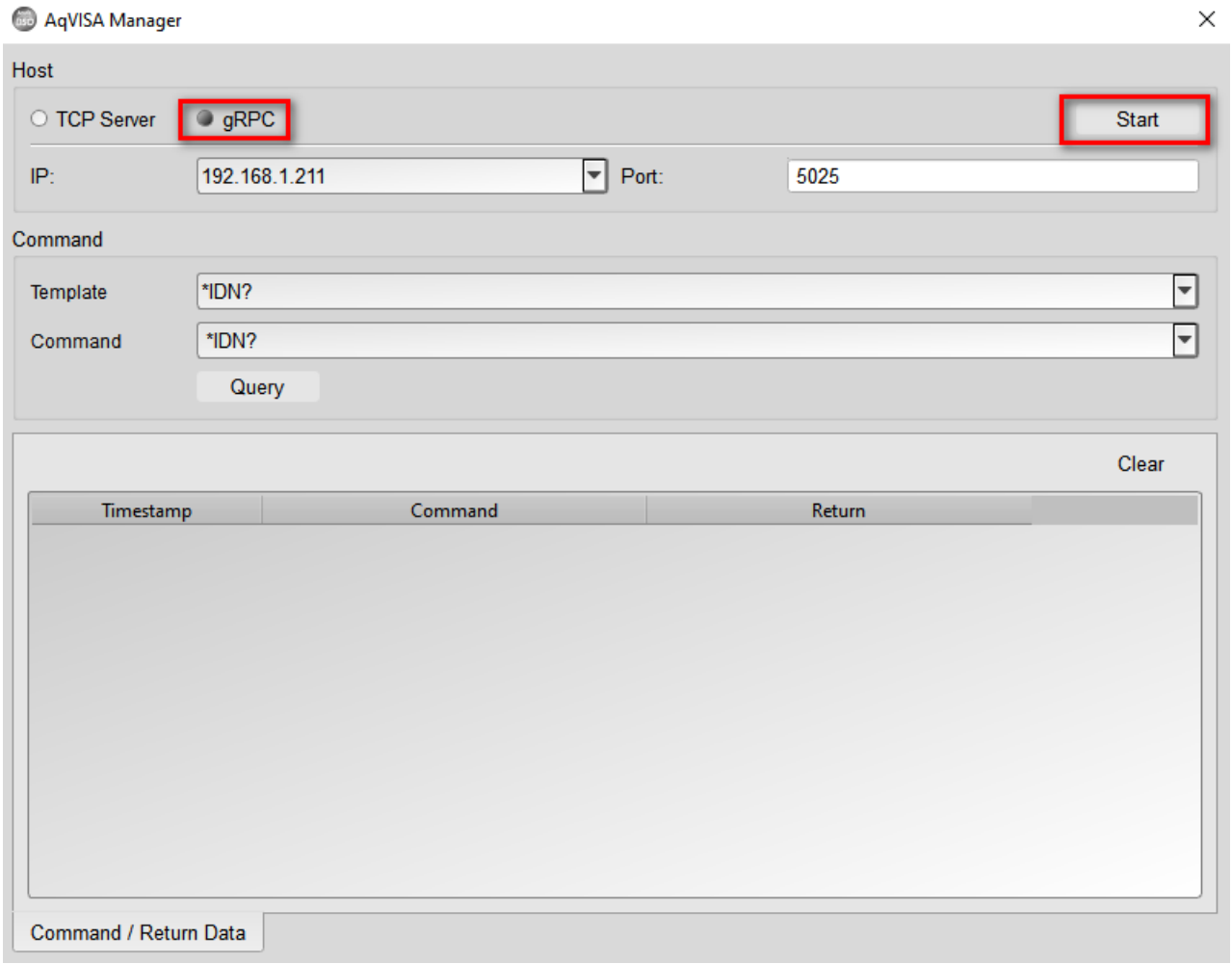


For DSO software, select **Utility > AqVISA Manager**.



When the host is set to **gRPC** and the **Start** is pressed, the gRPC server will start running on port 5025.

Copyright © 2025 Acute Technology Inc.



Or, you can start the server when launching the software in command line.

Take MSO as an example

Windows Cmd

```
MSO(x64).exe --host grpc --port 5025
```

Windows Powershell

```
.\MSO(x64).exe --host grpc --port 5025
```

Linux

```
./Mixed_Signal_Oscilloscope-x86_64.AppImage --host grpc --port 5025
```

3.2.2 Python Sample Client

We release a sample client code in Python. Follow the steps below to run the sample client. Please also refer to GitHub Repository for more details.

Copyright © 2025 Acute Technology Inc.

Requirements

We currently using gRPC v1.56.0.

```
pip install -r requirements.txt
```

Protobuf Complile

Please find `aqvisa.proto` in the `proto/` directory. Execute the following command to compile

```
python -m grpc_tools.protoc -Iproto
                            --python_out=.
                            --pyi_out=.
                            --grpc_python_out=.
                            proto/aqvisa.proto
```

For more information about protobuf, please visit Protocol Buffers official website.

Sample Codes

Please replace the `address` variable with your own ip and port first. You can obtain those values from the DSO or LA application.

VIWRITE & VIREAD

Simply run the code to make a single request to the application

```
python write_read.py
```

Add any custom commands in `command_list` to try out any commands you prefer.

VIWRITEFROMFILE & VIREADTOFILE

Additionally replace the `json_file` variable with a dedicated path. Then, simply run the code to send a JSON payload to the application.

```
python write_file.py
```

Besides, you are able to request a JSON config from the application.

```
python read_file.py
```


Copyright © 2025 Acute Technology Inc.

3.3 Protobuf Definition

3.3.1 Remote Procedure Calls (RPCs)

ViRead

Query response values if any from the previous requesting command sent by ViWrite.

See also ViReadRequest and ViReadResponse messages.

ViWrite

Send commands to the application with the parameter *command*. Return AQVI_NO_ERROR when successfully executed. Otherwise, the server responses a error code defined in AQVI_STATUS.

See also ViWriteRequest and ViWriteResponse messages.

ViWriteFromFile

This RPC is used for sending configuration file instead of sending commands to the application. The configuration file is in *JSON* format. It is loaded and the payload can be transferred over gRPC.

See also ViWriteFromFileRequest and ViWriteFromFileResponse messages.

For the *JSON* config, please see schema for more details.

3.3.2 Messages

ViReadRequest

- Parameter
- bytes ***jobid*** [deprecated]: NOT IN USE.
- int64 ***count***: Represents the size of the data buffer to receive data from the application.

ViReadResponse

- Parameter
- AQVI_STATUS ***status_code***: Return status.
- bytes ***command_response***: Queried response from the previous ViWrite command.
- int64 ***ret_count***: Represents the length of the responses

ViWriteRequest

- Parameter
- bytes ***command***: A byte array command that is ready to be sent to the application.

ViWriteResponse

- Parameter
- bytes ***jobid***: Represents the unique identification of the viwrite call.
- AQVI_STATUS ***status_code***: Return status.

Copyright © 2025 Acute Technology Inc.

ViWriteFromFileRequest

- Parameter
- bytes ***payload***: A byte array payload loaded from JSON configuration file. Currently supported schema is as follows

Schema Name	Schema
Electrical Validation	Schema

ViWriteFromFileResponse

- Parameter
- bytes ***jobid***: Represents the unique identification of the ViWriteFromFile call.
- AQVI_STATUS ***status_code***: Return status.

ViReadToFileRequest

- Parameter
- AQVI_JSON_SCHEMA ***schema_type***: Logical index of schema type
- int64 ***count***: Input buffer size for accepting requested payload Currently supported schema is as follows

Schema Name	Schema
Electrical Validation	Schema

ViReadToFileResponse

- Parameter
- AQVI_STATUS ***status_code***: Return status.
- bytes ***payload***: Requested JSON payload
- int64 ***ret_count***: Requested JSON payload size

3.3.3 Enums

AQVI_JSON_SCHEMA

Value	Enum	Description
0	AQVI_JSON_SCHEMA_DUMMY	Not in use
1	AQVI_JSON_SCHEMA_ELECTRICAL_VALIDATION	Electrical Validation Configuration

AQVI_STATUS

Same usage as status code in status code table.

Copyright © 2025 Acute Technology Inc.

4. Supported Commands

4.1 Common Commands

4.1.1 Command Formats

The following table is the list of common commands.

Command	Description
*IDN?	Query device information
*STB?	Query current software status

Copyright © 2025 Acute Technology Inc.

4.1.2 Command Definitions

*IDN?

Queries device information.

- **Syntax**:

```
*IDN?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  The format of response will be like

  Model: <Model Name>, Serial No.: <Device Serial Number>;

  If it is disconnected, then the response will be fixed as

  Model: Demo, Serial No.: Demo;

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*IDN?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Device information = %s\n", buffer);
```

*STB?

Queries current software status. Must be used after certain commands to check process completion.

Copyright © 2025 Acute Technology Inc.

Required after commands such as:

- *LA:CAPTURE:STOP
- *LA:DECODEREPORT:SAVE
- *LA:TIMINGREPORT:LOADSETTING
- *LA:TIMINGREPORT:SAVE
- *PA:CAPTURE:STOP
- *PA:REPORT:SAVE
- **Syntax**:

```
*STB?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  `0`: Software is busy `1`: Software is waiting for instructions

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*STB?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Software status = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

4.2 File Subsystem

4.2.1 Command Formats

The following table is the command tree of FILE subsystem.

Command	Description
*FILE	-
:LOAD	Load waveform file
:LOADLOG	A subset of command to load waveform logger file
:CONVERT	Starts converting a .LOG file.
:ENDTIME?	Returns the end time of the .LOG file.
:SPLITCNT?	Returns the number of splits in the .LOG file.
:STARTTIME?	Returns the start time of the .LOG file.
:SAVE	Save waveform file
:SAVEANALOG	Save analog waveform data (MSO series only)
:SAVEIMAGE	Save software screenshot to file
:SELECTPAGE	Select SubWindow page
:CLOSE	Close SubWindow page

4.2.2 Command Definitions

***FILE:LOAD**

Loads a waveform file

- **Syntax**:

```
*FILE:LOAD <filename>
```

- **Parameter**:

  <filename>: The path to the waveform file to be loaded.

Application	Supported File Extension
MSO	.MSW
BusFinder and Logic Analyzer	.BFW
TravelLogic	.TLW
TravelBus	.TBW

- **Example**:

```
*FILE:LOAD "SampleWaveform.MSW"
```


Copyright © 2025 Acute Technology Inc.

*FILE:LOADLOG:CONVERT

Converts a waveform logger file to a waveform file

- **Syntax**:

```
*FILE:LOADLOG:CONVERT <filename> <output_filename> {-INDEX <index>}
```

- **Parameter**:

  <filename>: The path to the waveform logger file to be loaded.

  <output_filename>: The path to the output waveform file.

  -INDEX <index> (optional): The index of the waveform to be loaded. If not specified, full range of waveforms will be converted.

- **viRead() Return Values**:

  The width and height of the application window.

- **Example**:

  Assume that a logger file (.LOG) is ready to be split into 3 smaller waveform files:

```
// Convert a logger file with full range (in this case include 0 - 2)
*FILE:LOADLOG:CONVERT "SampleWaveform.LOG" "SampleWaveform.MSW"
```

```
// Convert a logger file with index 1
*FILE:LOADLOG:CONVERT "SampleWaveform.LOG" "SampleWaveform.MSW" -INDEX 1
```

*FILE:LOADLOG:ENDTIME?

Returns the end time of the .LOG file.

- **Syntax**:

```
*FILE:LOADLOG:ENDTIME? <filename>
```

- **Parameter**:

  <filename>: The path to the waveform logger file to be loaded.

- **viRead() Return Values**:

  The end time of the .LOG file.

- **Example**:

```
*FILE:LOADLOG:ENDTIME? "SampleWaveform.LOG"
```


Copyright © 2025 Acute Technology Inc.

*FILE:LOADLOG:SPLITCNT?

Returns the number of splits in the .LOG file.

- **Syntax**:

```
*FILE:LOADLOG:SPLITCNT? <filename>
```

- **Parameter**:

  <filename>: The path to the waveform logger file to be loaded.

- **viRead() Return Values**:

  The number of splits in the .LOG file.

- **Example**:

```
*FILE:LOADLOG:SPLITCNT? "SampleWaveform.LOG"
```

*FILE:LOADLOG:STARTTIME?

Returns the start time of the .LOG file.

- **Syntax**:

```
*FILE:LOADLOG:STARTTIME? <filename>
```

- **Parameter**:

  <filename>: The path to the waveform logger file to be loaded.

- **viRead() Return Values**:

  The start time of the .LOG file.

- **Example**:

```
*FILE:LOADLOG:STARTTIME? "SampleWaveform.LOG"
```


Copyright © 2025 Acute Technology Inc.

*FILE:SAVE

Saves a waveform file.

- **Syntax**:

```
// Save a waveform file
*FILE:SAVE -WAVEFILE <filename>
```

```
// Save a transitional text file
*FILE:SAVE -TRANSTEXT <filename> [-NOTS] [-NOHEADER] [-TIMESEC]
```

```
// Save a text file, file size might significant larger than -TRANSTEXT
*FILE:SAVE -SAMPLETEXT <filename> [-NOTS] [-NOHEADER] [-TIMESEC]
```

- **Parameter**:

  -WAVEFILE <filename>: The path to the waveform file to be saved.

  -TRANSTEXT <filename>: The path to the transitional text file to be saved.

  -SAMPLETEXT <filename>: The path to the text file to be saved.

  -NOTS (optional): Do not save timestamp.

  -NOHEADER (optional): Do not save header.

  -TIMESEC (optional): Save timestamp in the unit of seconds.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save a waveform file as .MSW
*FILE:SAVE -WAVEFILE "SampleWaveform.MSW"
```

```
// Save a transitional text file as .TXT
*FILE:SAVE -TRANSTEXT "SampleWaveform.TXT"
```

```
// Save a text file as .TXT without timestamp
*FILE:SAVE -SAMPLETEXT "SampleWaveform.TXT" -NOTS
```


Copyright © 2025 Acute Technology Inc.

*FILE:SAVEANALOG

Saves analog waveform data (MSO series only).

- **Syntax**:

```
*FILE:SAVEANALOG <filename> -SAVEONFILE -CHSEL <channel_bits> [-NOTS]
```

```
*FILE:SAVEANALOG <filename> -SAVEEACHFILE -CHSEL <channel_bits> [-NOTS]
```

- **Parameter**:

  <filename>: The path to the waveform file to be saved.

  -SAVEONFILE: Save the waveform data to a single file.

  -SAVEEACHFILE: Save the waveform data to a file for each channel.

  -CHSEL <channel_bits>: Input -CHSEL with decimal BIT OR channel select flag to select target output channels. E.g., input -CHSEL 31 to output analog data from CH0 to CH4.

  -NOTS (optional): Do not save timestamp.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save CH0 and CH1 data with timestamp information into one file.
// Output file: File.csv
viWrite("*FILE:SAVEANALOG File.csv -SAVEONEFILE -CHSEL 3");
```

```
// Save CH1 and CH2 data without timestamp information into 2 files.
// Output files: File_DSO CH1.csv and File_DSO CH2.csv
viWrite("*FILE:SAVEANALOG File.csv -SAVEEACHFILE -CHSEL 6 -NOTS");
```


Copyright © 2025 Acute Technology Inc.

*FILE:SAVEIMAGE

Saves software screenshot to file.

- **Syntax**:

```
*FILE:SAVEIMAGE <filename>
```

- **Parameter**:

  <filename>: The path to the image file to be saved.

  Supported file extensions: .png, .jpg, .bmp

- **viRead() Return Values**:

  None

- **Example**:

```
// Save as BMP file
*FILE:SAVEIMAGE "SampleImage.bmp"
```

```
// Save as JPG file
*FILE:SAVEIMAGE "SampleImage.jpg"
```

```
// Save as PNG file
*FILE:SAVEIMAGE "SampleImage.png"
```

*FILE:SELECTPAGE

Selects a page of the waveform file.

- **Syntax**:

```
*FILE:SELECTPAGE <page_number>
```

- **Parameter**:

  <page_number>: The SubWindow page number to be selected. The first page index start from 0.

- **viRead() Return Values**:

  None

- **Example**:

```
// Select the first SubWindow page
viWrite("*FILE:SELECTPAGE 0");
```

```
// Select the 3rd SubWindow page
viWrite("*FILE:SELECTPAGE 2");
```


Copyright © 2025 Acute Technology Inc.

*FILE:CLOSE

Closes the SubWindow page

- **Syntax**:

```
// Close all SubWindow pages
*FILE:CLOSE -ALL
```

```
// Close the current SubWindow page
*FILE:CLOSE -CURRENT
```

- **Parameter**:

  None

- **Example**:

```
// Close current SubWindow page
viWrite("*FILE:CLOSE -CURRENT");
```

```
// Close all SubWindow pages
viWrite("*FILE:CLOSE -ALL");
```


Copyright © 2025 Acute Technology Inc.

4.3 Window Subsystem

4.3.1 Command Formats

The following table is the command tree of WINDOW subsystem.

Command	Description
*WINDOW	-
:SIZE	Sets application window size
:SIZE?	Query application window size
:SCREEN	Sets application window on screen index
:SCREEN?	Queries application window on screen index

4.3.2 Command Definitions

*WINDOW:SIZE

Sets application window size.

- **Syntax**:

```
*WINDOW:SIZE <width> <height>
```

- **Parameter**:

  <width>: Set application window width, e.g. 1000

  <height>: Set application window height, e.g. 800

- **Example**:

```
*WINDOW:SIZE 1000 800
```

*WINDOW:SIZE?

Queries application window size.

- **Syntax**:

```
*WINDOW:SIZE?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  The width and height of the application window.

- **Example**:

```
*WINDOW:SIZE?
```


Copyright © 2025 Acute Technology Inc.

*WINDOW:SCREEN

Sets application window on screen index.

- **Syntax**:

```
*WINDOW:SCREEN <screenIndex>
```

- **Parameter**:

  <screenIndex>: Set application window on screen index, e.g. 1

- **viRead() Return Values**:

  None

- **Example**:

```
*WINDOW:SCREEN 1
```

*WINDOW:SCREEN?

Queries application window on screen index.

- **Syntax**:

```
*WINDOW:SCREEN?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  The screen index of the application window.

- **Example**:

```
*WINDOW:SCREEN?
```


Copyright © 2025 Acute Technology Inc.

4.4 Logic Analyzer Mode Subsystem

4.4.1 What is Logic Analyzer Mode?

The **Logic Analyzer** mode allows you to capture and analyze digital signals. It provides specialized decoding and analysis features for various digital protocols like I2C, SPI, UART, and more. The Logic Analyzer mode helps engineers debug and verify digital communications by decoding raw signals into protocol-specific transactions and presenting them in an easy-to-understand format.

4.4.2 Command Formats

This section describes the command formats for the Logic Analyzer subsystem. Every command in the Logic Analyzer subsystem starts with ***LA:**.

Copyright © 2025 Acute Technology Inc.

The following table is the command tree of Logic Analyzer subsystem.

Command	Description
*LA:CAPTURE	A command set for capturing and analyzing protocol data
:START	Initiate waveform capture
:STOP	Stop waveform capture
:PROGRESS?	Read capture progress
:TRIGGERED?	Read capture trigger status
:STATUS?	Read capture status
*LA:CURSOR	A command set for Queriesing and saving report data
:DELETE	Delete selected cursor
:SEARCH	A subset of commands for cursor search
:SET	A subset of commands for setting cursor move/search condition
:BYEDGE	Set cursor move/search condition to edge transition
:BYTIME	Set cursor move/search condition to time period
:BYVALUEMATCH	Set cursor move/search condition to value or text match
:SEARCHPULSEWIDTH	Set cursor move/search condition to pulse width
:START	Start cursor move/search
*LA:WAVEFORM	A command set for Queriesing and saving report data
:SCALE	Queries report row count
:MOVETO	Queries report column count
:STORE	Save report to file
:RESTORE	Queries report data at selected index
*LA:WINDOWSIZE	A command set for Queriesing and saving report data
:WAVEFORMSIZE?	Queries the size of the waveform
:REPORTSIZE?	Queries the size of the report

Copyright © 2025 Acute Technology Inc.

Command	Description
*LA:DECODEREPORT	A command set for getting and saving decode report data
:COLUMNCOUNT?	Queries the amount of columns in the current selected decode report
:COUNT?	Queries the total number of reports
:DATA?	Queries report data at selected index
:NAME?	Queries the name of the current selected decode report
:ROWCOUNT?	Queries the amount of rows in the current selected decode report
:SAVE	Saves the current selected decode report to a file
:SELECT	Selects the index of the list of decode report
*LA:MEASUREMENTREPORT	A command set for getting and saving measurement report data
:COLUMNCOUNT?	Queries the amount of columns in the current selected measurement report
:COLUMNHEADER?	Queries the name of the column at the specified index
:DATA?	Queries the data at the specified row and column index
:REFRESH	Recalculates the measurement report
:ROWCOUNT?	Queries the amount of rows in the current selected measurement report
:SAVE	Saves the current selected measurement report to a file
*LA:TIMINGREPORT	A command set for getting and saving timing report data
:COLUMNCOUNT?	Queries the amount of columns in the current selected timing report
:COLUMNHEADER?	Queries the name of the column at the specified index
:DATA?	Queries the data at the specified row and column index
:LOADRESULT?	The result of the loading timing report process
:LOADSETTING	Loads the setting of the timing report
:ROWCOUNT?	Queries the amount of rows in the current selected timing report
:SAVE	Saves the current selected timing report to a file
:TESTRESULT?	The result of the test process
*LA:TRANSITIONREPORT	A command set for getting and saving transition report data
:COLUMNCOUNT?	Queries the amount of columns in the current selected transition report
:COLUMNHEADER?	Queries the name of the column at the specified index
:CURSORPOSITION?	Queries the position of the cursor
:DATA?	Queries the data at the specified row and column index
:ROWCOUNT?	Queries the amount of rows in the current selected transition report
:SAVE	Saves the current selected transition report to a file
:SELECT	Selects the index of the list of transition report

Copyright © 2025 Acute Technology Inc.

4.4.3 Command Definitions

*LA:CAPTURE:START

Initiate waveform capture.

- **Syntax**:

```
*LA:CAPTURE:START
```

- **Parameter**:

  None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_CAPTURE_ALREADY_RUNNING	Represents the acquisition is already in process.

- **viRead() Return Values**:

  None

- **Example**:

```
// Start logic analyzer acquistion
viWrite("*LA:CAPTURE:START");
```

*LA:CAPTURE:STOP

Stop waveform capture.

- **Syntax**:

```
*LA:CAPTURE:STOP
```

- **Parameter**: None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_CAPTURE_NOT_RUNNING	Represents that the application does not start the acquisition.

- **viRead() Return Values**:

  None

- **Example**:

```
// Stop logic analyzer acquistion
viWrite("*LA:CAPTURE:STOP");
```


Copyright © 2025 Acute Technology Inc.

*LA:CAPTURE:PROGRESS?

Read capture progress.

- **Syntax**:

```
*LA:CAPTURE:PROGRESS?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  A value that represents the current capture progress, which is in the range of 0.00 to 1.00.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:CAPTURE:PROGRESS?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Capture progress = %f\n", atof(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:CAPTURE:TRIGGERED?

Read capture trigger status.

- **Syntax**:

```
*LA:CAPTURE:TRIGGERED?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  `1`: The trigger event has occurred.

  `0`: The trigger event has not occurred.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:CAPTURE:PROGRESS?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Capture triggered = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

***LA:CAPTURE:STATUS?**

Read capture status.

Copyright © 2025 Acute Technology Inc.

- **Syntax**:

```
*LA:CAPTURE:STATUS?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  A string that represents the current capture status.

Status	Description
Data Transferring	The data is transferring.
Data Processing	The data is processing.
Capture Finished	The logic analyzer has finished capturing data.
Analyzing	The logic analyzer is analyzing the data.
Analysis Finished	The logic analyzer has finished analyzing the data.
Pre-Trigger	The logic analyzer enters pre-trigger state.
Wait For Trigger	The logic analyzer is waiting for trigger condition.
Post-Trigger	The logic analyzer enters post-trigger state.
Wait For DSO Capture	The logic analyzer is waiting for DSO capture.
Capturing	The logic analyzer is capturing data.
Saving Report	The logic analyzer is currently saving a report.
Report Save Finished	The logic analyzer has finished saving a report.
File Load Finished	The logic analyzer has finished loading a file.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:CAPTURE:STATUS?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Capture status = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:CURSOR:DELETE

Delete selected cursor.

- **Syntax**:

```
// Delete all cursors
*LA:CURSOR:DELETE -ALL
```

```
// Delete specified cursors
*LA:CURSOR:DELETE <cursorN>
```

- **Parameter**:

  -ALL: Delete all cursors.

  <cursorN>: Delete specified cursors.

- **viRead() Return Values**: None
- **Example**:

```
// Delete all cursors
viWrite("*LA:CURSOR:DELETE -ALL");
```

```
// Delete CursorB
viWrite("*LA:CURSOR:DELETE CursorB");
```

*LA:CURSOR:SEARCH:SET:BYEDGE

Set Cursor Move/Search condition to edge transition.

- **Syntax**:

```
*LA:CURSOR:SEARCH:SET:BYEDGE <label> <edgeCount> <edgeType>
```

- **Parameter**:

  <label>: Set target label, e.g. CH-04

  <edgeCount>: Set condition edge count.

  <edgeType>: Set condition edge type.

- **viRead() Return Values**: None
- **Example**:

```
// Set Cursor Move/Search condition to CH-05 Rising Edge x8
viWrite("*LA:CURSOR:SEARCH:SET:BYEDGE CH-05 8 RISE");
```

```
// Set Cursor Move/Search condition to CH-15 Falling Edge x10
viWrite("*LA:CURSOR:SEARCH:SET:BYEDGE CH-15 10 FALL");
```


Copyright © 2025 Acute Technology Inc.

*LA:CURSOR:SEARCH:SET:BYTIME

Set Cursor Move/Search condition to time period.

- **Syntax**:

```
*LA:CURSOR:SEARCH:SET:BYTIME <time>
```

- **Parameter**:

  <time>: Set time condition, e.g. 10ns, 50ms

- **viRead() Return Values**: None
- **Example**:

```
// Set cursor move / search condition to 50.5us
viWrite("*LA:CURSOR:SEARCH:SET:BYTIME 50.5us");
```

*LA:CURSOR:SEARCH:SET:BYVALUEMATCH

Set Cursor Move/Search condition to value or text match.

- **Syntax**:

```
*LA:CURSOR:SEARCH:SET:BYVALUEMATCH <label> <value>
```

- **Parameter**:

  <label>: Set target label, e.g. CH-04

  <value>: Set Value/Text match condition, e.g. 05h or "Data Read"

- **viRead() Return Values**: None
- **Example**:

```
// Set Cursor Move/Search condition to value 50h (Hexadecimal)
viWrite("*LA:CURSOR:SEARCH:SET:BYVALUEMATCH 50h");
```

```
// Set Cursor Move/Search condition to text match on "Data Read"
viWrite("*LA:CURSOR:SEARCH:SET:BYVALUEMATCH \"Data Read\"");
```


Copyright © 2025 Acute Technology Inc.

*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH

Set Cursor Move/Search condition to pulse width.

- **Syntax**:

```
*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH <label> <pulseType> G|L|GE|LE|E|NE <time>

*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH <label> <pulseType> INRANGE|OUTRANGE <time1> <time2>
```

- **Parameter**:

  <label>: Set target label, e.g. CH-04

  <pulseType>: Set pulse type to High, Low, or Either.

  <conditionType>:

```
- G: Greater than
- L: Less than
- GE: Greater than or equal to
- LE: Less than or equal to
- E: Equal to
- NE: Not equal to
- INRANGE: In range, requires two time conditions to be specified.
- OUTRANGE: Out of range, requires two time conditions to be specified.
```

  <time>: Set time condition, e.g. 10ns, 50ms

- **viRead() Return Values**: None
- **Example**:

```
// Set Cursor Move/Search condition to CH-03 positive pulse width to 10 us
viWrite("*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH CH-03 HIGH L 10us");
```

```
//Set Cursor Move/Search condition to CH-13 any pulse width between 1 ms and 10 ms
viWrite("*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH CH-13 ALL INRANGE 1ms 10ms");
```


Copyright © 2025 Acute Technology Inc.

*LA:CURSOR:SEARCH:START

Start Cursor Move/Search.

- **Syntax**:

```
*LA:CURSOR:SEARCH:START <direction> <cursor> [MULTIPLE <count>]
```

- **Parameter**:

  <direction>: Set search direction to Left or Right.

  <cursor>: Set target cursor, e.g. CursorA, CursorB, ... , CursorZ

  MULTIPLE <count>: Set moving multiple cursors, additional cursor move count needs to be specified, e.g. 5.

- **viRead() Return Values**: None
- **Example**:

```
// Starting from cursor C, searching and place cursor C to E to the right.
viWrite("*LA:CURSOR:SEARCH:START RIGHT CursorC MULTIPLE 3");
```

*LA:WAVEFORM:SCALE

Set the horizontal time scale of the waveform.

- **Syntax**:

```
*LA:WAVEFORM:SCALE <time>
```

- **Parameter**:

  time: Horizontal waveform time scale, units: ps, ns, us, ms, s.

- **viRead() Return Values**:

  None

- **Example**:

```
// Set the waveform time scale to 10 us.
viWrite("*LA:WAVEFORM:SCALE 10us");
```


Copyright © 2025 Acute Technology Inc.

*LA:WAVEFORM:MOVETO

Move waveform center to specified cursor position.

- **Syntax**:

```
*LA:WAVEFORM:MOVETO <cursor>
```

- **Parameter**:

  cursor: Cursor name, e.g., CursorA, CursorB, ... , CursorZ

- **viRead() Return Values**: None
- **Example**:

```
// Move the center point of the waveform to location of Cursor C.
viWrite("*LA:WAVEFORM:MOVETO CURSORC");
```

*LA:WAVEFORM:STORE

Store current waveform position or time scale.

- **Syntax**:

```
*LA:WAVEFORM:STORE [-SCALE] [-POSITION]
```

- **Parameter**:

  -*SCALE*: Store current waveform time scale.

  -*POSITION*: Store current waveform position.

- **viRead() Return Values**: None
- **Example**:

```
// Store current waveform position.
viWrite("*LA:WAVEFORM:STORE -POSITION");
```

```
// Store current waveform time scale.
viWrite("*LA:WAVEFORM:STORE -SCALE");
```

```
// Store both current waveform position and time scale.
viWrite("*LA:WAVEFORM:STORE -POSITION -SCALE");
```


Copyright © 2025 Acute Technology Inc.

*LA:WAVEFORM:RESTORE

Restore previous waveform position or time scale.

- **Syntax**:

```
*LA:WAVEFORM:RESTORE [-SCALE] [-POSITION]
```

- **Parameter**:

  -*SCALE*: Restore previous waveform time scale.

  -*POSITION*: Restore previous waveform position.

- **viRead() Return Values**: None
- **Example**:

```
// Restore previous waveform position.
viWrite("*LA:WAVEFORM:RESTORE -POSITION");
```

```
// Restore previous waveform time scale.
viWrite("*LA:WAVEFORM:RESTORE -SCALE");
```

```
// Restore both previous waveform position and time scale.
viWrite("*LA:WAVEFORM:RESTORE -POSITION -SCALE");
```

*LA:WINDOWSIZE:REPORTSIZE

Set report area size, will also change the waveform area size.

- **Syntax**:

```
*LA:WINDOWSIZE:REPORTSIZE <size>
```

- **Parameter**:

  SIZE: Specify report area size, range from 0 to 100 (unit: percentage).

- **viRead() Return Values**: None
- **Example**:

```
// Set report area size to 50% of the screen
viWrite("*LA:WINDOWSIZE:REPORTSIZE 50");
```


Copyright © 2025 Acute Technology Inc.

*LA:WINDOWSIZE:WAVEFORMSIZE

Set waveform area size, will also change the report area size.

- **Syntax**:

```
*LA:WINDOWSIZE:WAVEFORMSIZE <size>
```

- **Parameter**:

  SIZE: Specify waveform area size, range from 0 to 100 (unit: percentage).

- **viRead() Return Values**: None
- **Example**:

```
// Set waveform area size to 50% of the screen
viWrite("*LA:WINDOWSIZE:WAVEFORMSIZE 50");
```

*LA:DECODEREPORT:COLUMNCOUNT?

Get current Decode report column count

- **Syntax**:

```
*LA:DECODEREPORT:COLUMNCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Current Decode report column count

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:DECODEREPORT:COLUMNCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Decode report column count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:DECODEREPORT:COUNT?

Get Decode report count

- **Syntax**:

```
*LA:DECODEREPORT:COUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Current Decode report count

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:DECODEREPORT:NAME?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Decode report count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:DECODEREPORT:DATA?

Get Decode report data at selected index.

- **Syntax**:

```
*LA:DECODEREPORT:DATA? <row> <column>
```

- **Parameter**:

  row: Decode report row index.

  column: Decode report column index.

- **viRead() Return Values**:

  Decode report data at the selected index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:DECODEREPORT:DATA? 2 1")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

printf("Decode report data at (2, 1) = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:DECODEREPORT:NAME?

Get current Decode report name

- **Syntax**:

```
*LA:DECODEREPORT:NAME?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Current Decode report name

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:DECODEREPORT:NAME?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Decode report name = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:DECODEREPORT:ROWCOUNT?

Get current Decode report row count

- **Syntax**:

```
*LA:DECODEREPORT:ROWCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Current Decode report row count

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:DECODEREPORT:ROWCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Decode report row count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:DECODEREPORT:SAVE

Save Decode report

- **Syntax**:

```
*LA:DECODEREPORT:SAVE <filename> [-BEGIN <row>] [-END <row>] [-NOTS] [-USESEC] [-NOHEADER]
```

- **Parameter**:

  filename: Target filename. It can be either a relative path or an absolute path.

  -BEGIN <row> (optional): Input the report save start row, default starts at begin of the report.

  -END <row> (optional): Input the report save end row, default ends at end of the report.

  -NOTS (optional): The report save feature stores timestamp information at the first column of the saved report, add this parameter to specify NOT store timestamp information into the saved report.

  -USESEC (optional): The report save feature will store timestamp information in nano-seconds in the saved report, add this parameter to specify using seconds to store the timestamp information.

  -NOHEADER (optional): Specify NOT to include the header information in the saved report.

- **viRead() Return Values**:

  None

- **Example**:

```
//Save report to MyDecodeReport.csv
viWrite("*LA:DECODEREPORT:SAVE MyDecodeReport.csv");
```

> **Note**
>
> Be sure that your are able to write to the destination file directory.

*LA:DECODEREPORT:SELECT

Select decode report.

- **Syntax**:

```
*LA:DECODEREPORT:SELECT <index>
```

- **Parameter**:

  index: Decode report index.

- **viRead() Return Values**:

  None

- **Example**:

```
// Select the first decode report
viWrite("*LA:DECODEREPORT:SELECT 0");
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:COLUMNCOUNT?

Get Measurement Report column count.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:COLUMNCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return current Measurement Report column count.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:MEASUREMENTREPORT:COLUMNCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Measurement report column count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:COLUMNHEADER?

Get Measurement Report column header name at specified column index.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:COLUMNHEADER? <column>
```

- **Parameter**:

  column: Specify column index.

- **viRead() Return Values**:

  Return current Measurement Report column header name at specified column index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:MEASUREMENTREPORT:COLUMNHEADER? 1")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Measurement report column header at index 1 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:DATA?

Get Measurement Report data at specified row and column index.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:DATA? <row> <column>
```

- **Parameter**:

  row: Specify row index.

  column: Specify column index.

- **viRead() Return Values**:

  Measurement Report data at specified row and column index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:MEASUREMENTREPORT:DATA? 1 2")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Measurement report data at [1, 2] = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:REFRESH

Recalculate Measurement Report.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:REFRESH
```

- **Parameter**:

  None

- **viRead() Return Values**:

  None

- **Example**:

```
// Recalculate Measurement Report
viWrite("*LA:MEASUREMENTREPORT:REFRESH");
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:ROWCOUNT?

Get Measurement Report row count.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:ROWCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return current Measurement Report row count.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:MEASUREMENTREPORT:ROWCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Measurement report row count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:MEASUREMENTREPORT:SAVE

Save Measurement Report to file.

- **Syntax**:

```
*LA:MEASUREMENTREPORT:SAVE <filename> [-BEGIN <row>] [-END <row>] [-NOTS] [-USESEC] [-NOHEADER]
```

- **Parameter**:

  filename: Target filename. It can be either a relative path or an absolute path.

  -BEGIN <row> (optional): Input the report save start row, default starts at begin of the report.

  -END <row> (optional): Input the report save end row, default ends at end of the report.

  -NOTS (optional): The report save feature stores timestamp information at the first column of the saved report, add this parameter to specify NOT store timestamp information into the saved report.

  -USESEC (optional): The report save feature will store timestamp information in nano-seconds in the saved report, add this parameter to specify using seconds to store the timestamp information.

  -NOHEADER (optional): Specify NOT to include the header information in the saved report.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save Measurement Report as .csv file to MyMeasurementReport.csv
viWrite("*LA:MEASUREMENTREPORT:SAVE MyMeasurementReport.csv");
```

```
// Save Measurement Report as .html file to MyMeasurementReport.html
viWrite("*LA:MEASUREMENTREPORT:SAVE MyMeasurementReport.html");
```

> **Note**
>
> Be sure that your are able to write to the destination file directory.

Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:COLUMNCOUNT?

Get Timing Report column count.

- **Syntax**:

```
*LA:TIMINGREPORT:COLUMNCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Current Timing Report column count.

- Example

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:COLUMNCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report column count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:COLUMNHEADER?

Get Timing Report column header.

- **Syntax**:

```
*LA:TIMINGREPORT:COLUMNHEADER? <column>
```

- **Parameter**:

  column: Timing report column index.

- **viRead() Return Values**:

  Current Timing Report column header.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:COLUMNHEADER? 1")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report column header at index 1 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:DATA?

Get Timing Report data at specified row and column index.

- **Syntax**:

```
*LA:TIMINGREPORT:DATA? <row> <column>
```

- **Parameter**:

  row: Timing report row index.

  column: Timing report column index.

- **viRead() Return Values**:

  Timing Report data at specified row and column index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:DATA? 1 2")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report data at [1, 2]= %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:LOADRESULT?

Load Timing Report result.

- **Syntax**:

```
*LA:TIMINGREPORT:LOADRESULT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return the process result of *LA:TIMINGREPORT:LOADSETTING, return "OK" if no error, otherwise, return error information.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:LOADRESULT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report load result = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:LOADSETTING

Load Timing Report settings.

- **Syntax**:

```
*LA:TIMINGREPORT:LOADSETTING <filename>
```

- **Parameter**:

  filename: Timing report setting filename.

- **viRead() Return Values**:

  None

- **Example**:

```
// Load Timing Report setting
viWrite("*LA:TIMINGREPORT:LOADSETTING MyTimingSettings.csv");
```

> **Note**
>
> Be sure that your are able to write to the destination file directory.

Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:ROWCOUNT?

Get Timing Report row count.

- **Syntax**:

```
*LA:TIMINGREPORT:ROWCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return current Timing Report row count.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:ROWCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report row count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:SAVE

Save Timing Report to file.

- **Syntax**:

```
// Save timing report as .csv file
*LA:TIMINGREPORT:SAVE -CSV <filename>
```

```
// Save timing report as .txt file
*LA:TIMINGREPORT:SAVE -TXT <filename>
```

```
// Save timing report as .html file
*LA:TIMINGREPORT:SAVE -HTML <filename> [-TITLE <title>] [-USERINPUT <userInput>]
```

- **Parameter**:

  filename: Target filename. It can be either a relative path or an absolute path.

  -CSV: Save report into .csv file.

  -TXT: Save report into .txt file.

  -HTML: Save report into .html file.

  -TITLE <title> (optional): Specify HTML report title.

  -USERINPUT <userInput> (optional): Specify user input information.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save timing report as .csv file to MyTimingReport.csv
viWrite("*LA:TIMINGREPORT:SAVE –CSV MyTimingReport.csv");
```

```
// Save timing report as .html file to MyTimingReport.html with a title
viWrite("*LA:TIMINGREPORT:SAVE –HTML MyTimingReport.html –TITLE My Example Title");
```

> **Note**
>
> Be sure that your are able to write to the destination file directory.

Copyright © 2025 Acute Technology Inc.

*LA:TIMINGREPORT:TESTRESULT?

Get overall Pass/Fail result from Timing Report.

- **Syntax**:

```
*LA:TIMINGREPORT:TESTRESULT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return `Pass` if all items are tested passed, otherwise, return `Fail`.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TIMINGREPORT:TESTRESULT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Timing report test result = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:COLUMNCOUNT?

Get Transition Report column count.

- **Syntax**:

```
*LA:TRANSITIONREPORT:COLUMNCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return current Transition Report column count.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TRANSITIONREPORT:COLUMNCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Transition report column count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:COLUMNHEADER?

Get Transition Report column header name at specified column index.

- **Syntax**:

```
*LA:TRANSITIONREPORT:COLUMNHEADER? <column>
```

- **Parameter**:

  <column>: Specify column index.

- **viRead() Return Values**:

  Return current Transition Report column header name at specified column index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TRANSITIONREPORT:COLUMNHEADER? 1")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Transition report column header name at [1] = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:DATA?

Get Transition Report data at specified row and column index.

- **Syntax**:

```
*LA:TRANSITIONREPORT:DATA? <row> <column>
```

- **Parameter**:

  row: Specify row index.

  column: Specify column index.

- **viRead() Return Values**:

  Transition Report data at specified row and column index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TRANSITIONREPORT:DATA? 1 2")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Transition report data at [1, 2] = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:ROWCOUNT?

Get Transition Report row count.

- **Syntax**:

```
*LA:TRANSITIONREPORT:ROWCOUNT?
```

- **Parameter**:

  None

- **viRead() Return Values**:

  Return current Transition Report row count.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TRANSITIONREPORT:ROWCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Transition report row count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:SAVE

Save Transition Report to file.

- **Syntax**:

```
*LA:TRANSITIONREPORT:SAVE <filename> [-BEGIN <row>] [-END <row>] [-NOTS] [-USESEC] [-NOHEADER]
```

- **Parameter**:

  filename: Target filename. It can be either a relative path or an absolute path.

  -BEGIN <row> (optional): Input the report save start row, default starts at begin of the report.

  -END <row> (optional): Input the report save end row, default ends at end of the report.

  -NOTS (optional): The report save feature stores timestamp information at the first column of the saved report, add this parameter to specify NOT store timestamp information into the saved report.

  -USESEC (optional): The report save feature will store timestamp information in nano-seconds in the saved report, add this parameter to specify using seconds to store the timestamp information.

  -NOHEADER (optional): Specify NOT to include the header information in the saved report.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save transition report as .csv file to MyTransitionReport.csv
viWrite("*LA:TRANSITIONREPORT:SAVE –CSV MyTransitionReport.csv");
```

```
// Save transition report as .html file to MyTransitionReport.html
viWrite("*LA:TRANSITIONREPORT:SAVE –HTML MyTransitionReport.html");
```

> **Note**
>
> Be sure that your are able to write to the destination file directory.

Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:SELECT

Select Label and refresh Transition report.

- **Syntax**:

```
// Select all labels
*LA:TRANSITIONREPORT:SELECT -ALL
```

```
// Select specific labels
*LA:TRANSITIONREPORT:SELECT <label1> <label2> ... <labelN>
```

- **Parameter**:

  -ALL: Select all labels.

  <labelN>: Select specific labels.

- **viRead() Return Values**:

  None

- **Example**:

```
//Select Label and refresh Transition report
viWrite("*LA:TRANSITIONREPORT:SELECT");
```

```
//Select all labels to create Transition Report

viWrite("*LA:TRANSITIONREPORT:SELECT -ALL");
```

```
//Select label CH-00, Bus[7:0] and CH50 to create Transition Report
viWrite("*LA:TRANSITIONREPORT:SELECT CH-00 Bus[7:0] CH50");
```


Copyright © 2025 Acute Technology Inc.

*LA:TRANSITIONREPORT:CURSORPOSITION?

Get row number of the Transition Report where the specified Cursor is located.

- **Syntax**:

```
*LA:TRANSITIONREPORT:CURSORPOSITION? <cursor>
```

- **Parameter**:

  <cursor>: Specify Cursor name, e.g., CursorA, CursorB, ... , CursorZ

- **viRead() Return Values**:

  Return current Transition Report row number where the specified Cursor is located.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*LA:TRANSITIONREPORT:CURSORPOSITION? CursorA")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Transition report cursor position = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

4.5 Protocol Analyzer Mode Subsystem

4.5.1 What is Protocol Analyzer Mode?

The **Protocol Analyzer** mode allows you to capture and analyze serial protocol data. It provides specialized decoding and analysis features for various serial protocols like I2C, SPI, UART, and more. The Protocol Analyzer mode helps engineers debug and verify serial communications by decoding raw signals into protocol-specific transactions and presenting them in an easy-to-understand format.

Key capabilities include:

- Real-time decoding of protocol signals
- Detailed transaction views with statistics
- Data export and reporting

4.5.2 Command Formats

This section describes the command formats for the Protocol Analyzer subsystem. Every command in the Protocol Analyzer subsystem starts with ***PA:**.

The following table is the command tree of Protocol Analyzer subsystem.

Command	Description
*PA:CAPTURE	A command set for capturing and analyzing protocol data
:START	Initiate waveform capture
:STOP	Stop waveform capture
*PA:REPORT	A command set for querying and saving report data
:ROWCOUNT?	Get report row count
:COLUMNCOUNT?	Get report column count
:SAVE	Save report to file
:DATA?	Get report data at selected index

Copyright © 2025 Acute Technology Inc.

4.5.3 Command Definitions

*PA:CAPTURE:START

Start Protocol Analyzer capture.

- **Syntax**:

```
*PA:CAPTURE:START
```

- **Parameter**:

  None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_CAPTURE_ALREADY_RUNNING	Represents the acquisition is already in process.

- **viRead() Return Values**:

  None

- **Example**:

```
// Start protocol analyzer acquistion
viWrite("*PA:CAPTURE:START");
```

*PA:CAPTURE:STOP

Stop Protocol Analyzer capture.

- **Syntax**:

```
*PA:CAPTURE:STOP
```

- **Parameter**:

  None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_CAPTURE_NOT_RUNNING	Represents that the application does not start the acquisition.

- **Query Values**:

  None

- **Example**:

```
// Stop protocol analyzer acquistion
viWrite("*PA:CAPTURE:STOP");
```


Copyright © 2025 Acute Technology Inc.

*PA:REPORT:ROWCOUNT?

Retrieve the number of rows of the report.

- **Syntax**:

```
*PA:REPORT:ROWCOUNT?
```

- **Parameter**:

  None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.

- **viRead() Return Values**:

  The number of rows of the report.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*PA:REPORT:ROWCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Row count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*PA:REPORT:COLUMNCOUNT?

Retrieve the number of columns of the report.

- **Syntax**:

```
*PA:REPORT:COLUMNCOUNT?
```

- **Parameter**:

  None

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.

- **viRead() Return Values**:

  The number of columns of the report.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*PA:REPORT:COLUMNCOUNT?")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Column count = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

*PA:REPORT:SAVE

Saves the report to the file with the specified filename using designated type.

- **Syntax**:

```
*PA:REPORT:SAVE <filename> [-SAVEALL] [-ADVRPT] [-SPLITDUR]
```

```
*PA:REPORT:SAVE <filename> [-SAVESPLIT <row>] [-ADVRPT] [-SPLITDUR]
```

```
*PA:REPORT:SAVE <filename> [-SAVERANGE <start row> <end row> <start column> <end column>] [-ADVRPT]
[-SPLITDUR]
```

- **Parameter**:

  filename: Target filename.

  -SAVEALL (optional): Saves the entire report into one file.

  -ADVRPT (optional): Includes the advanced report data.

  -SPLITDUR (optional): Separates the report timestamp and duration data into two columns.

  -SAVESPLIT <row> (optional): Splits the report into multiple files based on row count.

  -SAVERANGE (optional): Saves the report within selected row and column range.

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_INPUT_FILE_DIR_INVALID	Represents the input filename is invalid.

- **viRead() Return Values**:

  None

- **Example**:

```
// Save the entire report into one file
viWrite("\*PA:REPORT:SAVE C:/Temp/AllReport.csv -SAVEALL");
```

```
// Save the entire report into one file with advanced report data
viWrite("\*PA:REPORT:SAVE C:/Temp/AdvReport.csv -SAVEALL -ADVRPT");
```

```
// Split the report into multiple files based on row count = 2000
viWrite("\*PA:REPORT:SAVE C:/Temp/LimitRpt.csv -SAVESPLIT 2000");
```

```
// Save the report within selected row and column range
viWrite("\*PA:REPORT:SAVE C:/Temp/RangedRpt.csv -SAVERANGE 200 2000 1 5");
```


Copyright © 2025 Acute Technology Inc.

*PA:REPORT:DATA?

Retrieve report data from the selected index.

- **Syntax**:

```
*PA:REPORT:DATA? <row> <column>
```

- **Parameter**:

  row: The row index.

  column: The column index.

- **Status Codes**:

Status Code	Description
AQVI_NO_ERROR	Command executed successfully.
AQVI_ROW_COL_INDEX_INVALID	Represents the column index is invalid.

- **viRead() Return Values**:

  The report data at the selected index.

- **Example**:

```
// Send command with viWrite()
if (!viWrite("*PA:REPORT:DATA? 105 2")) {
    return; // viWrite failed
}

// Wait for software process command
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// Check if the command is executed successfully
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // No returned data or process command failed
    return;
}

// Fetch response from application buffer with viRead()
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead failed
}

// Display the result
printf("Report data at index [105, 2] = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

Copyright © 2025 Acute Technology Inc.

5. Error Codes

5.1 Status Code Table

Status Code	Value	Description
AQVI_NO_ERROR	0	Operation succeeded without error
AQVI_ALREADY_OPEN	1	OPENRM should only be called once to establish the software connection
AQVI_APPLICATION_NOT_STARTED	2	The main software application was not started before calling the viOPENRM
AQVI_FAILED_TO_CONNECT	3	Failed to establish the connection to software application
AQVI_CONNECTION_NOT_VALID	4	The software connection is not valid yet, call viOpenRM first to establish the connection
AQVI_CONNECTION_LOST	5	The software application has been closed or re-started, need to call viCloseRM and viOpenRM to establish the connection again
AQVI_FAILED_TO_COMMUNICATE	6	Unable to send data to software application
AQVI_NO_RETURN_DATA	7	No data was returned from the software application
AQVI_INPUT_NULL_POINTER	8	Input data buffer is NULL pointer and not valid
AQVI_DATA_BUFFER_TOO_SMALL	9	Input data buffer size is too small
AQVI_SEL_APP_AFTER_VM_OPENED	10	Select application type should be done before OpenVM
AQVI_SEL_APP_TYPE_INVALID	11	Selected invalid application type
AQVI_PREVIOUS_CMD_PROCESSING	12	Previous input command still processing
AQVI_INPUT_PARAMETER_UNKNOWN	13	Input parameter unknown or not supported
AQVI_INPUT_PARAMETER_INCOMPLETED	14	Input parameter is incomplete
AQVI_TIMEOUT	15	Input request timeout

Copyright © 2025 Acute Technology Inc.

Status Code	Value	Description
AQVI_NOT_SUPPORTED	1000	Not supported in current software
AQVI_INCOMPLETE_COMMAND	1001	Input command incomplete
AQVI_SUBWND_INVALID	1002	Input command requires an existing SubWindow
AQVI_SUBWND_CNT_EXCEED	1003	Input command attempts to add new SubWindow page, but software has already exceeded maximum page count
AQVI_SW_BUSY	1004	Input command ignored while software busy
AQVI_LASUBWND_INVALID	1005	Input command requires exist Logic Analyzer SubWindow
AQVI_PASUBWND_INVALID	1006	Input command requires exist Protocol Analyzer SubWindow
AQVI_DECODE_REPORT_INVALID	1007	Input command requires exist Decode Report
AQVI_TIMING_REPORT_INVALID	1008	Input command requires exist Timing Report
AQVI_INPUT_COMMAND_FORMAT_ERROR	1009	Input command format error
AQVI_INPUT_FILE_DIR_INVALID	1010	Input file directory invalid
AQVI_CAPTURE_ALREADY_RUNNING	1011	Sending Capture Start command while capture already running
AQVI_CAPTURE_NOT_RUNNING	1012	Sending Capture Stop command while capture is not running
AQVI_ROW_COL_INDEX_INVALID	1013	Input Row or Column index invalid
AQVI_SELECT_INDEX_INVALID	1014	Input index selection invalid
AQVI_INPUT_PARAMETER_INVALID	1015	Missing input parameters or not enough input parameter
AQVI_INPUT_SETTING_FILE_FORMAT_ERROR	1016	Input setting file format error
AQVI_FILE_ACCESS_ERROR	1017	Unable to access File, unable to reach select file directory or selected unsupported file type
AQVI_TRANSITION_REPORT_INVALID	1018	Input command requires an existing Transition Report
AQVI_MEASUREMENT_REPORT_INVALID	1019	Input command requires exist Measurement Report
AQVI_SELECT_LABEL_INVALID	1020	Selected Label Name not valid
AQVI_SELECT_RANGE_ERROR	1021	Selected Range Condition Error, can't mix Line and Cursor Range Condition
AQVI_SELECT_CHANNEL_NAME_INVALID	1022	Selected Channel Name in Cursor Search is not valid
AQVI_CURSOR_SEARCH_FAILED	1023	Cursor Search Failed, unable to find more valid match
AQVI_WAVEFORM_NOT_READY	1024	Waveform Data Not Ready
AQVI_NOT_IN_EV_MODE	1025	Software is not configured in EV mode
AQVI_NO_EV_ANALYSIS_RESULT	1026	Software doesn't have any valid EV analysis result
AQVI_SELECT_DEV_NOT_EXIST	1027	Selected device S/N not present in device list
AQVI_UNSUPPORT_FEATURE	9995	Unsupported feature, feature is not supported in current model or demo mode
AQVI_UNFINISHED_FEATURE	9996	Unfinished Feature

Copyright © 2025 Acute Technology Inc.

Status Code	Value	Description
AQVI_UNKNOWN_COMMAND_ERROR	9997	Input command is not supported
AQVI_UNKNOWN_ERROR	9998	Detected unknown error
AQVI_DLL_NOT_READY	9999	DLL initialization not finished, please retry again later

Copyright © 2025 Acute Technology Inc.

6. Release Notes

Version	Date	Description
1.0.9	2025-03-09	Update the documentation
1.0.9 rev. 2	2025-03-20	Fix a discrepancy in the documentation with the saving file commands
1.0.9 rev. 3	2025-03-21	Add installation instructions

Copyright © 2025 Acute Technology Inc.