# 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 概述 9
2.2.2 安装步骤说明 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 启动 gRPC 服务器 13
3.2.2 Python 示例客户端 14
3.3 Protobuf Definition 16
3.3.1 远程过程调用 (RPCs) 16
3.3.2 Message 16
3.3.3 Enum 17
4. Supported Commands 18
4.1 Common Commands 18
4.1.1 命令格式 18
4.1.2 命令定义 19
4.2 File Subsystem 21
4.2.1 命令格式 21


Copyright © 2025 Acute Technology Inc.

4.2.2 命令定义 21
4.3 Window Subsystem 28
4.3.1 命令格式 28
4.3.2 命令定义 28
4.4 Logic Analyzer Mode Subsystem 30
4.4.1 什么是逻辑分析仪模式？ 30
4.4.2 命令格式 30
4.4.3 命令定义 33
4.5 Protocol Analyzer Mode Subsystem 73
4.5.1 什么是协议分析仪模式？ 73
4.5.2 命令格式 73
4.5.3 命令定义 74
5. Error Codes 80
5.1 状态码表 80
6. Release Notes 83


Copyright © 2025 Acute Technology Inc.

# 1. Introduction

## 1.1 Scope

这是 Acute LAVISA（简称 AqLAVISA）的完整文档。

本文档包含以下主题：

- AqLAVISA 基本概念和功能
- AqLAVISA 提供的接口及其使用方法
- 支持的命令及其使用方法

本文档不包含以下主题：

- 第三方应用程序的系统设计细节
- 特定协议的解码和分析细节

AqLAVISA 提供了一个直观的接口，用于无缝管理和控制 Acute 的逻辑分析仪。开发人员可以使用此接口来捕获、分析和保存来自逻辑分析仪的数据。


Copyright © 2025 Acute Technology Inc.

## 1.2 Core Concepts

AqLAVISA 提供了 API 接口，允许用户程序通过文本命令与 Acute 逻辑分析仪软件进行通信。用户可以通过这个接口访问协议分析仪页面中的协议分析报告、解码报告或逻辑分析仪页面中的时序分析报告。

对于自动化测试，开发人员可以通过加载示例波形文件来设置捕获和环境设置，然后开始捕获以记录新数据，并读取解码或分析信息来检查和验证目标设备的行为。

AqLAVISA 接口允许开发人员通过发送纯文本命令来实现软件自动化。开发人员能够在采集过程后收集解码信息、时序测量等数据。 最初，这个接口是通过共享库（.dll）公开的，该库处理所有接收到的命令并与软件进行通信。然而，这种方式存在一个问题，即它仅限于物理连接仪器的单台机器本地使用。因此，远程控制成为一种使其更加通用和灵活的方式。


Copyright © 2025 Acute Technology Inc.

# 1.3 Supported Platforms

## 1.3.1 Windows

| 软件 | 版本 |
| --- | --- |
| MSO | 1.7.82 或更高 |
| MSO (Full) | 1.7.82 或更高 |
| BusFinder and Logic Analyzer | 1.7.74 或更高 |
| TravelLogic | 1.7.47 或更高 |
| TravelBus | 1.7.34 或更高 |

## 1.3.2 Linux

**Note**

Linux 平台上的应用程序仍标记为测试版。如果您在运行程序时遇到任何错误，请创建问题报告。

| 软件 | 版本 |
| --- | --- |
| Mixed_Signal_Oscilloscope | 2.0.0 或更高 |
| Digital_Storage_Oscilloscope | 2.0.0 或更高 |
| BusFinder_Logic_Analyzer | 2.0.0 或更高 |
| TravelLogic_Analyzer | 2.0.0 或更高 |
| TravelBus_Analyzer | 2.0.0 或更高 |


Copyright © 2025 Acute Technology Inc.

## 1.4 Interfaces

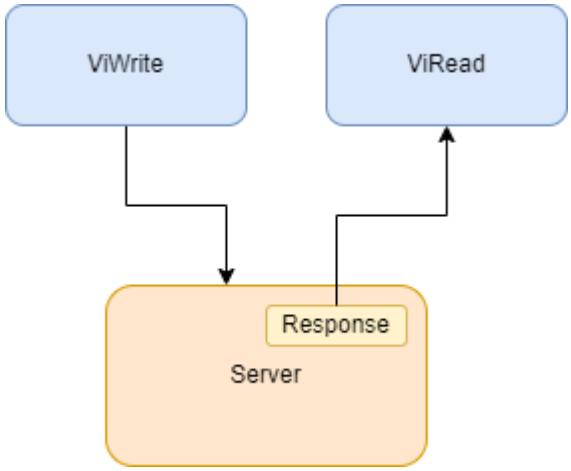
AqLAVISA 提供了三种不同的接口来与逻辑分析仪进行交互。每个接口都有相关的使用说明、教程和 API 参考链接。支持以下接口：

- 本地套接字进程间通信: 仅限 Windows 平台
- TCP 套接字连接: 仅限 Windows 平台
- gRPC


Copyright © 2025 Acute Technology Inc.

# 2. Basic Control

## 2.1 Flow Chart




只需通过 viWrite 调用发送命令即可。服务器将在命令完全执行后生成响应。如果有需要返回的查询值，您可以使用 viRead 调用来获取该值。


Copyright © 2025 Acute Technology Inc.

# 2.2 Installation

## 2.2.1 概述

AqVISA 包含以下几个部分：

- 动态链接库（DLL）文件
- 64 位 Windows 系统使用 `AqVISA64.dll`
- 库依赖文件

| 组件 | 文件名 |
| --- | --- |
| Qt 5 核心库 | `Qt5Core.dll` |
| Qt 5 网络库 | `Qt5Network.dll` |
| ICU 库 | `icudt.dll`、`icuin.dll`、`icuuc.dll` |
| C++ 运行时库 | `msvcp120.dll`、`msvcr120.dll` |

- 用户手册
- `AqVISA_UserManual_chs.pdf`
- Python 用户的 Python 包

  PyPI 包：aqvisa GitHub 仓库：aqvisa-python

## 2.2.2 安装步骤说明

- 运行安装程序

  请参考安装页面下载安装程序。 默认情况下，安装程序会将动态链接库和依赖库安装到以下目录：

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

  安装程序提供将安装目录添加到系统 PATH 的选项。 建议大多数用户选择此选项，以便轻松找到动态链接库。

-（可选）安装 Python 包

  ```
  您可以使用名为 `pip` 的包管理器从 Python 包索引（PyPI）安装 aqvisa 包。

  ```bash
  pip install aqvisa
  ```
  ```

- 示例使用

  请参考 GitHub 仓库中的示例代码。

  Python


Copyright © 2025 Acute Technology Inc.

## 2.3 DLL Functions

### 2.3.1 viSelectAppType

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

设置目标连接的Acute逻辑分析仪软件。

**参数**

- `iAppType [In]`:

  类型：***int***

  可用的参数如下列表所示

| 参数 | 值 | 目标软件 | 建议软件版本 |
|---|---|---|---|
| APP_TYPE_TL | 0 | TL (x64).exe / TL.exe | 1.7.47 or higher |
| APP_TYPE_BF | 1 | BF_LA3K(x64).exe / BF_LA3K.exe | 1.7.74 or higher |
| APP_TYPE_TB | 2 | TBA (x64).exe / TBA.exe | 1.7.34 or higher |
| APP_TYPE_MS | 3 | MSO (x64).exe / MSO.exe | 1.7.82 or higher |

**回传值**

成功时返回1；否则返回0。

### 2.3.2 viOpenRM

```
int viOpenRM()
```

连接到逻辑分析仪软件。

**回传值**

成功时返回1；否则返回0。

### 2.3.3 viWrite

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

向逻辑分析仪软件发送AqLAVISA指令。

**参数**

- `szCommand [In]`:

  类型：***char ****

  包含AqLAVISA指令，详细信息请参考Acute LAVISA指令列表。

**回传值**

成功时返回1；否则返回0。


Copyright © 2025 Acute Technology Inc.

### 2.3.4 viRead

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

从逻辑分析仪软件读取AqLAVISA指令执行结果。

**参数**

- `szReturnBuffer [Out]`:

  类型：***char ****

  用于AqLAVISA指令返回值的数据缓冲区。

- `iBufferLength [In]`:

  类型：***int***

  用于szReturnBuffer的缓冲区大小。

**回传值**

返回读取到的字符长度。

### 2.3.5 viGetCommandResult

```
int viGetCommandResult()
```

读取前一个AqLAVISA viWrite函数调用的指令处理结果。

**回传值**

| 返回值 | 值 | 描述 |
| --- | --- | --- |
| PROC_OK_WOUT_RETURN | 0 | 处理成功且无返回数据。 |
| PROC_OK_WITH_RETURN | 1 | 处理成功且有返回数据。 |
| PROC_FAIL | 2 | 处理失败。 |
| PROC_PENDING | 3 | 处理中。 |

### 2.3.6 viCloseRM

```
int viCloseRM()
```

停用AqLAVISA功能并断开与逻辑分析仪软件的连接。

**回传值**

成功时返回1；否则返回0。

### 2.3.7 viErrCode

```
int viErrCode()
```

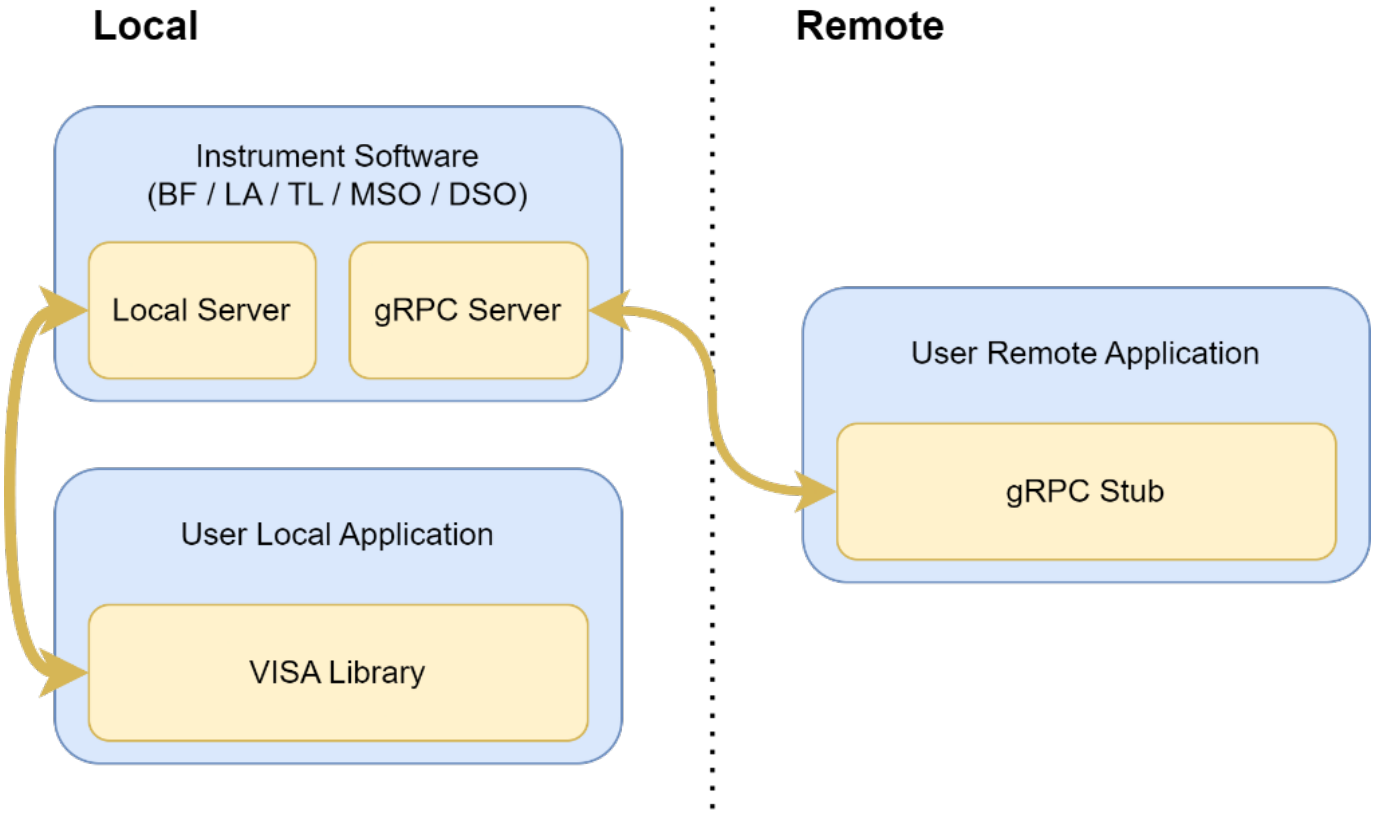
获取AqLAVISA错误代码。详细信息请参考错误码。


Copyright © 2025 Acute Technology Inc.

# 3. Remote Control

## 3.1 Overview

与许多自动化系统一样，AqVISA 通过允许新类型的文本命令来保持接口的一致性并增强其功能。它整体上围绕着客户端-服务器模型的理念。在服务器端，仪器软件实现了 AqVISA 接口并运行服务器来处理传入的命令。从下面的框图中我们可以看到，软件运行本地服务器和 gRPC 服务器来处理来自本地应用程序或远程应用程序的请求。




在客户端，由于 gRPC 支持多语言特性，开发人员可以使用他们喜欢的任何编程语言构建应用程序。例如，您可以轻松地用 Python、C# 等语言创建客户端应用程序。

在本节中，我们将只介绍通过 gRPC 进行远程控制的设置。

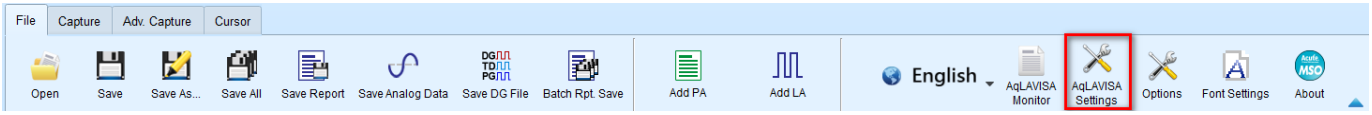

Copyright © 2025 Acute Technology Inc.

# 3.2 Quick Start

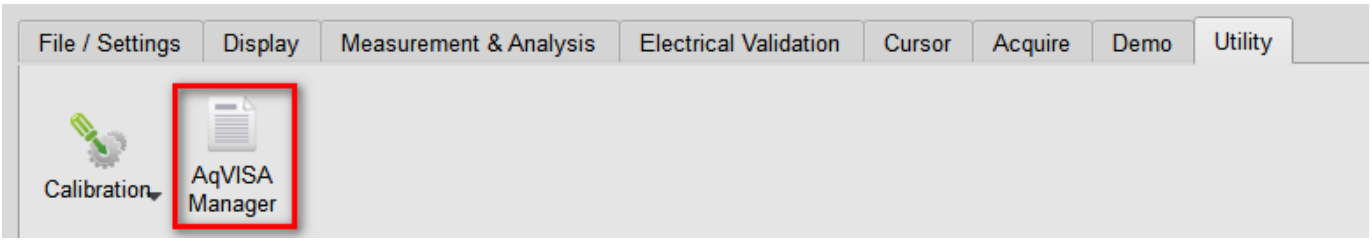
## 3.2.1 启动 gRPC 服务器

gRPC 服务器可以通过软件界面或命令行启用

对于 BusFinder / Logic Analyzer / TravelLogic / MSO 软件，选择 **文件 > AqLAVISA 管理器**。

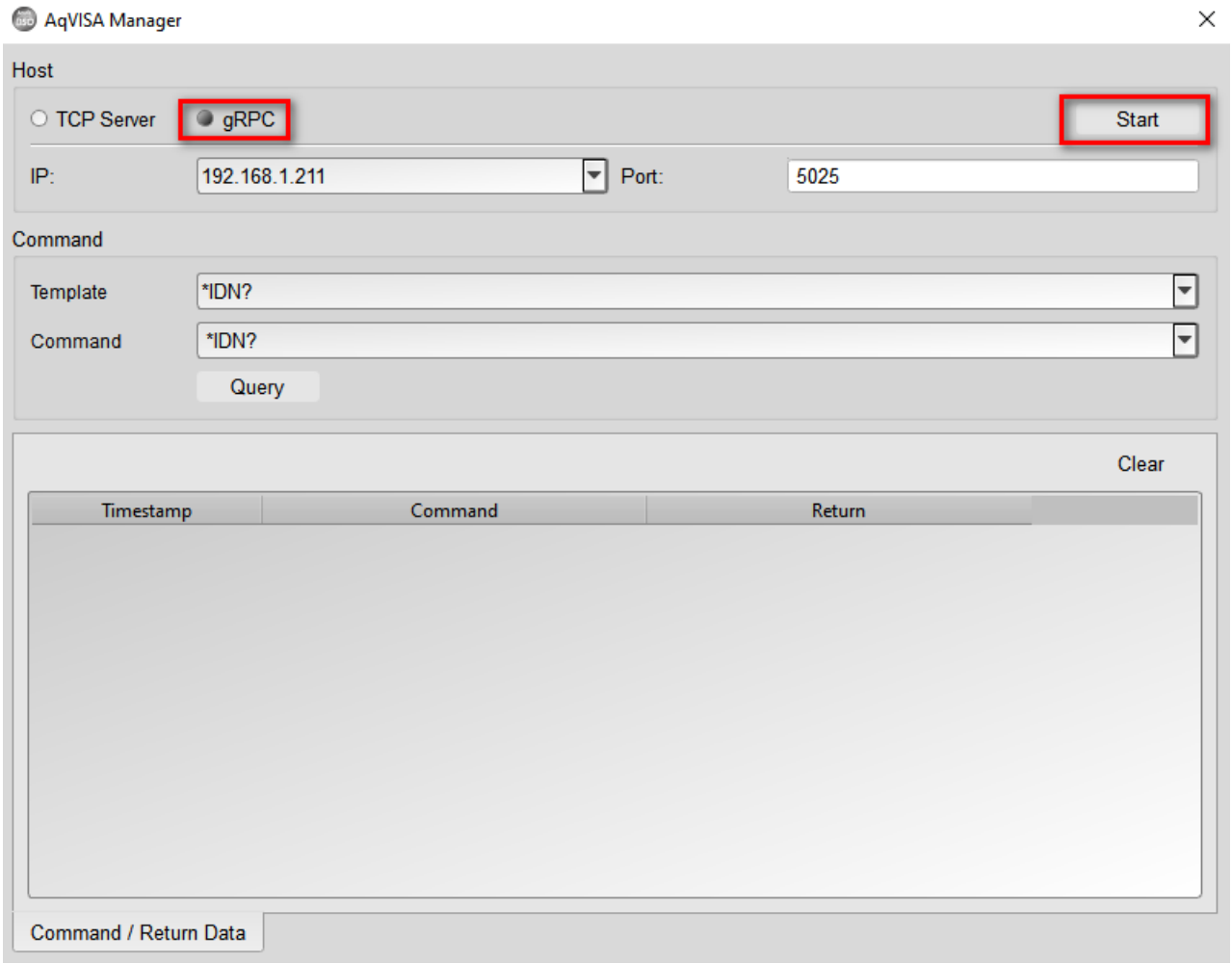



对于 DSO 软件，选择 **实用工具 > AqVISA 管理器**。




当主机设置为 **gRPC** 并按下 **启动** 时，gRPC 服务器将在端口 5025 上开始运行。


Copyright © 2025 Acute Technology Inc.




或者，您可以在命令行中启动软件时启动服务器。

以 MSO 为例

Windows 命令提示符

```
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 示例客户端

我们发布了一个 Python 示例客户端代码。按照以下步骤运行示例客户端。更多详细信息请参考 GitHub 仓库。


Copyright © 2025 Acute Technology Inc.

#### 要求

我们目前使用 gRPC v1.56.0。

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

#### Protobuf 编译

请在 `proto/` 目录中找到 `aqvisa.proto`。执行以下命令进行编译

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

有关 protobuf 的更多信息，请访问 Protocol Buffers 官方网站。

#### 示例代码

请首先将 `address` 变量替换为您自己的 IP 和端口。您可以从 DSO 或 LA 应用程序中获取这些值。

##### VIWRITE 和 VIREAD

简单运行代码向应用程序发送单个请求

```
python write_read.py
```

在 `command_list` 中添加任何自定义命令来尝试您想要的任何命令。

##### VIWRITEFROMFILE 和 VIREADTOFILE

另外将 `json_file` 变量替换为指定路径。然后，简单运行代码向应用程序发送 JSON 负载。

```
python write_file.py
```

此外，您也可以从应用程序请求 JSON 配置。

```
python read_file.py
```


Copyright © 2025 Acute Technology Inc.

## 3.3 Protobuf Definition

### 3.3.1 远程过程调用 (RPCs)

#### ViRead

查询上一个由 ViWrite 发送的命令的响应值（如果有）。

另请参见 ViReadRequest 和 ViReadResponse 消息。

#### ViWrite

使用参数 *command* 向应用程序发送命令。成功执行时返回 AQVI_NO_ERROR。否则，服务器返回在 AQVI_STATUS 中定义的错误代码。

另请参见 ViWriteRequest 和 ViWriteResponse 消息。

#### ViWriteFromFile

此 RPC 用于发送配置文件而不是向应用程序发送命令。配置文件采用 *JSON* 格式。它被加载后可以通过 gRPC 传输。

另请参见 ViWriteFromFileRequest 和 ViWriteFromFileResponse 消息。

关于 *JSON* 配置，请参见 schema 获取更多详细信息。

### 3.3.2 Message

#### ViReadRequest

- 参数
- bytes ***jobid*** [已弃用]：不再使用。
- int64 ***count***：表示用于从应用程序接收数据的数据缓冲区大小。

#### ViReadResponse

- 参数
- AQVI_STATUS ***status_code***：返回状态。
- bytes ***command_response***：来自上一个 ViWrite 命令的查询响应。
- int64 ***ret_count***：表示响应的长度

#### ViWriteRequest

- 参数
- bytes ***command***：准备发送到应用程序的字节数组命令。

#### ViWriteResponse

- 参数
- bytes ***jobid***：表示 viwrite 调用的唯一标识。
- AQVI_STATUS ***status_code***：返回状态。


Copyright © 2025 Acute Technology Inc.

#### ViWriteFromFileRequest

- 参数
- bytes ***payload***：从 JSON 配置文件加载的字节数组负载。 当前支持的架构如下：

| 架构名称 | 架构 |
| --- | --- |
| 电气验证 | 架构 |

#### ViWriteFromFileResponse

- 参数
- bytes ***jobid***：表示 ViWriteFromFile 调用的唯一标识。
- AQVI_STATUS ***status_code***：返回状态。

#### ViReadToFileRequest

- 参数
- AQVI_JSON_SCHEMA ***schema_type***：架构类型的逻辑索引
- int64 ***count***：用于接受请求负载的输入缓冲区大小 当前支持的架构如下：

| 架构名称 | 架构 |
| --- | --- |
| 电气验证 | 架构 |

#### ViReadToFileResponse

- 参数
- AQVI_STATUS ***status_code***：返回状态。
- bytes ***payload***：请求的 JSON 负载
- int64 ***ret_count***：请求的 JSON 负载大小

### 3.3.3 Enum

#### AQVI_JSON_SCHEMA

| 值 | Enum | 描述 |
| --- | --- | --- |
| **0** | AQVI_JSON_SCHEMA_DUMMY | 未使用 |
| **1** | AQVI_JSON_SCHEMA_ELECTRICAL_VALIDATION | 电气验证配置 |

#### AQVI_STATUS

用法与 状态码表 中的状态码相同。


Copyright © 2025 Acute Technology Inc.

# 4. Supported Commands

## 4.1 Common Commands

### 4.1.1 命令格式

下表是常用命令列表。

| 命令 | 描述 |
| --- | --- |
| *IDN? | 查询设备信息 |
| *STB? | 查询当前软件状态 |


Copyright © 2025 Acute Technology Inc.

## 4.1.2 命令定义

***IDN?**

查询设备信息。

- **语法**:

```
*IDN?
```

- **参数**:

  无

- **viRead() 返回值**:

  响应格式如下

  *Model: <型号名称>, Serial No.: <设备序列号>;*

  如果设备未连接，则响应将固定为

  *Model: Demo, Serial No.: Demo;*

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*IDN?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否成功执行
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("设备信息 = %s\n", buffer);
```

***STB?**

查询当前软件状态。必须在某些命令之后使用以检查处理是否完成。



Copyright © 2025 Acute Technology Inc.

需要在以下命令之后使用：

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

```
*STB?
```

- **参数**:

  无

- **viRead() 返回值**:

  `0`：软件正忙 `1`：软件等待指令

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*STB?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否成功执行
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("软件状态 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

# 4.2 File Subsystem

## 4.2.1 命令格式

下表是 FILE 子系统的命令树。

| 命令 | 描述 |
| --- | --- |
| *FILE | - |
| :LOAD | 加载波形文件 |
| :LOADLOG | 加载波形日志文件的子命令集 |
| :CONVERT | 开始转换 .LOG 文件 |
| :ENDTIME? | 返回 .LOG 文件的结束时间 |
| :SPLITCNT? | 返回 .LOG 文件的分段数量 |
| :STARTTIME? | 返回 .LOG 文件的开始时间 |
| :SAVE | 保存波形文件 |
| :SAVEANALOG | 保存模拟波形数据（仅 MSO 系列） |
| :SAVEIMAGE | 保存软件截图到文件 |
| :SELECTPAGE | 选择子窗口页面 |
| :CLOSE | 关闭子窗口页面 |

## 4.2.2 命令定义

***FILE:LOAD**

加载波形文件

- **语法**:

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

- **参数**:

  *<filename>*: 要加载的波形文件路径。

| 应用程序 | 支持的文件扩展名 |
| --- | --- |
| MSO | .MSW |
| BusFinder and Logic Analyzer | .BFW |
| TravelLogic | .TLW |
| TravelBus | .TBW |

- **示例**:

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


Copyright © 2025 Acute Technology Inc.

***FILE:LOADLOG:CONVERT**

将波形日志文件转换为波形文件

- **语法**:

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

- **参数**:

  *<filename>*: 要加载的波形日志文件路径。

  *<output_filename>*: 输出波形文件的路径。

  *-INDEX <index> (可选)*: 要加载的波形索引。如果未指定，将转换全部范围的波形。

- **viRead() 返回值**:

  应用程序窗口的宽度和高度。

- **示例**:

  假设一个日志文件（.LOG）准备被分割成 3 个较小的波形文件：

```
// 转换完整范围的日志文件（在本例中包括 0 - 2）
*FILE:LOADLOG:CONVERT "SampleWaveform.LOG" "SampleWaveform.MSW"
```

```
// 转换索引为 1 的日志文件
*FILE:LOADLOG:CONVERT "SampleWaveform.LOG" "SampleWaveform.MSW" -INDEX 1
```

***FILE:LOADLOG:ENDTIME?**

返回 .LOG 文件的结束时间。

- **语法**:

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

- **参数**:

  *<filename>*: 要加载的波形日志文件路径。

- **viRead() 返回值**:

  .LOG 文件的结束时间。

- **示例**:

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



Copyright © 2025 Acute Technology Inc.

#### *FILE:LOADLOG:SPLITCNT?

返回 .LOG 文件的分段数量。

- **语法**:

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

- **参数**:

  *<filename>*: 要加载的波形日志文件路径。

- **viRead() 返回值**:

  .LOG 文件的分段数量。

- **示例**:

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

#### *FILE:LOADLOG:STARTTIME?

返回 .LOG 文件的开始时间。

- **语法**:

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

- **参数**:

  *<filename>*: 要加载的波形日志文件路径。

- **viRead() 返回值**:

  .LOG 文件的开始时间。

- **示例**:

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


Copyright © 2025 Acute Technology Inc.

***FILE:SAVE**

保存波形文件。

- **语法**:

```
// 保存波形文件
*FILE:SAVE -WAVEFILE <filename>
```

```
// 保存过渡文本文件
*FILE:SAVE -TRANSTEXT <filename> [-NOTS] [-NOHEADER] [-TIMESEC]
```

```
// 保存文本文件，文件大小可能比 -TRANSTEXT 大很多
*FILE:SAVE -SAMPLETEXT <filename> [-NOTS] [-NOHEADER] [-TIMESEC]
```

- **参数**:

  *-WAVEFILE <filename>*: 要保存的波形文件路径。

  *-TRANSTEXT <filename>*: 要保存的过渡文本文件路径。

  *-SAMPLETEXT <filename>*: 要保存的文本文件路径。

  *-NOTS (可选)*: 不保存时间戳。

  *-NOHEADER (可选)*: 不保存头信息。

  *-TIMESEC (可选)*: 以秒为单位保存时间戳。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将波形文件保存为 .MSW
*FILE:SAVE -WAVEFILE "SampleWaveform.MSW"
```

```
// 将过渡文本文件保存为 .TXT
*FILE:SAVE -TRANSTEXT "SampleWaveform.TXT"
```

```
// 将文本文件保存为 .TXT，不包含时间戳
*FILE:SAVE -SAMPLETEXT "SampleWaveform.TXT" -NOTS
```


Copyright © 2025 Acute Technology Inc.

***FILE:SAVEANALOG**

保存模拟波形数据（仅 MSO 系列）。

- **语法**:

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

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

- **参数**:

  *<filename>*: 要保存的波形文件路径。

  *-SAVEONFILE*: 将波形数据保存到单个文件。

  *-SAVEEACHFILE*: 为每个通道保存一个波形数据文件。

  *-CHSEL <channel_bits>*: 输入 -CHSEL 与十进制位 OR 通道选择标志来选择目标输出通道。例如，输入 -CHSEL 31 以输出 CH0 到 CH4 的模拟数据。

  *-NOTS (可选)*: 不保存时间戳。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将 CH0 和 CH1 的数据及时间戳信息保存到一个文件中。
// 输出文件：File.csv
viWrite("*FILE:SAVEANALOG File.csv -SAVEONEFILE -CHSEL 3");
```

```
// 将 CH1 和 CH2 的数据（不含时间戳信息）保存到 2 个文件中。
// 输出文件：File_DSO CH1.csv 和 File_DSO CH2.csv
viWrite("*FILE:SAVEANALOG File.csv -SAVEEACHFILE -CHSEL 6 -NOTS");
```


Copyright © 2025 Acute Technology Inc.

#### *FILE:SAVEIMAGE

保存软件截图到文件。

- **语法**:

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

- **参数**:

  *<filename>*: 要保存的图像文件路径。

  支持的文件扩展名：.png、.jpg、.bmp

- **viRead() 返回值**:

  无

- **示例**:

```
// 保存为 BMP 文件
*FILE:SAVEIMAGE "SampleImage.bmp"
```

```
// 保存为 JPG 文件
*FILE:SAVEIMAGE "SampleImage.jpg"
```

```
// 保存为 PNG 文件
*FILE:SAVEIMAGE "SampleImage.png"
```

#### *FILE:SELECTPAGE

选择波形文件的页面。

- **语法**:

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

- **参数**:

  *<page_number>*: 要选择的子窗口页面编号。第一个页面索引从 0 开始。

- **viRead() 返回值**:

  无

- **示例**:

```
// 选择第一个子窗口页面
viWrite("*FILE:SELECTPAGE 0");
```

```
// 选择第三个子窗口页面
viWrite("*FILE:SELECTPAGE 2");
```


Copyright © 2025 Acute Technology Inc.

***FILE:CLOSE**

关闭子窗口页面

- **语法**:

```
// 关闭所有子窗口页面
*FILE:CLOSE -ALL
```

```
// 关闭当前子窗口页面
*FILE:CLOSE -CURRENT
```

- **参数**:

  无

- **示例**:

```
// 关闭当前子窗口页面
viWrite("*FILE:CLOSE -CURRENT");
```

```
// 关闭所有子窗口页面
viWrite("*FILE:CLOSE -ALL");
```


Copyright © 2025 Acute Technology Inc.

# 4.3 Window Subsystem

## 4.3.1 命令格式

下表是 WINDOW 子系统的命令树。

| 命令 | 描述 |
| --- | --- |
| *WINDOW | - |
| :SIZE | 设置应用程序窗口大小 |
| :SIZE? | 查询应用程序窗口大小 |
| :SCREEN | 设置应用程序窗口在指定屏幕索引上 |
| :SCREEN? | 查询应用程序窗口所在的屏幕索引 |

## 4.3.2 命令定义

***WINDOW:SIZE**

设置应用程序窗口大小。

- **语法**:

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

- **参数**:

  *<width>*: 设置应用程序窗口宽度，例如 1000

  *<height>*: 设置应用程序窗口高度，例如 800

- **示例**:

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

***WINDOW:SIZE?**

查询应用程序窗口大小。

- **语法**:

```
*WINDOW:SIZE?
```

- **参数**:

  无

- **viRead() 返回值**:

  应用程序窗口的宽度和高度。

- **示例**:

```
*WINDOW:SIZE?
```



Copyright © 2025 Acute Technology Inc.

#### *WINDOW:SCREEN

设置应用程序窗口在指定屏幕索引上。

- **语法**:

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

- **参数**:

  *<screenIndex>*: 设置应用程序窗口在指定屏幕索引上，例如 1

- **viRead() 返回值**:

  无

- **示例**:

```
*WINDOW:SCREEN 1
```

#### *WINDOW:SCREEN?

查询应用程序窗口所在的屏幕索引。

- **语法**:

```
*WINDOW:SCREEN?
```

- **参数**:

  无

- **viRead() 返回值**:

  应用程序窗口所在的屏幕索引。

- **示例**:

```
*WINDOW:SCREEN?
```


Copyright © 2025 Acute Technology Inc.

# 4.4 Logic Analyzer Mode Subsystem

## 4.4.1 什么是逻辑分析仪模式？

**逻辑分析仪**模式允许您采集和分析数字信号。 它为各种数字协议（如 I2C、SPI、UART 等）提供专门的解码和分析功能。 逻辑分析仪模式通过将原始信号解码为特定协议的事务并以易于理解的格式呈现，帮助工程师调试和验证数字通信。

## 4.4.2 命令格式

本节描述逻辑分析仪子系统的命令格式。逻辑分析仪子系统中的每个命令都以 ***LA:** 开头。


Copyright © 2025 Acute Technology Inc.

下表是逻辑分析仪子系统的命令树。

| 命令 | 描述 |
| --- | --- |
| *LA:CAPTURE | 用于采集和分析协议数据的命令集 |
| :START | 启动波形采集 |
| :STOP | 停止波形采集 |
| :PROGRESS? | 读取采集进度 |
| :TRIGGERED? | 读取采集触发状态 |
| :STATUS? | 读取采集状态 |
| *LA:CURSOR | 用于查询和保存报告数据的命令集 |
| :DELETE | 删除选定的光标 |
| :SEARCH | 光标搜索的子命令集 |
| :SET | 设置光标移动/搜索条件的子命令集 |
| :BYEDGE | 将光标移动/搜索条件设置为边沿跳变 |
| :BYTIME | 将光标移动/搜索条件设置为时间周期 |
| :BYVALUEMATCH | 将光标移动/搜索条件设置为值或文本匹配 |
| :SEARCHPULSEWIDTH | 将光标移动/搜索条件设置为脉冲宽度 |
| :START | 开始光标移动/搜索 |
| *LA:WAVEFORM | 用于查询和保存报告数据的命令集 |
| :SCALE | 查询报告行数 |
| :MOVETO | 查询报告列数 |
| :STORE | 保存报告到文件 |
| :RESTORE | 查询选定索引处的报告数据 |
| *LA:WINDOWSIZE | 用于查询和保存报告数据的命令集 |
| :WAVEFORMSIZE? | 查询波形的大小 |
| :REPORTSIZE? | 查询报告的大小 |


Copyright © 2025 Acute Technology Inc.

| 命令 | 描述 |
| --- | --- |
| *LA:DECODEREPORT | 用于获取和保存解码报告数据的命令集 |
| :COLUMNCOUNT? | 查询当前选定解码报告的列数 |
| :COUNT? | 查询报告总数 |
| :DATA? | 查询选定索引处的报告数据 |
| :NAME? | 查询当前选定解码报告的名称 |
| :ROWCOUNT? | 查询当前选定解码报告的行数 |
| :SAVE | 将当前选定的解码报告保存到文件 |
| :SELECT | 选择解码报告列表的索引 |
| *LA:MEASUREMENTREPORT | 用于获取和保存测量报告数据的命令集 |
| :COLUMNCOUNT? | 查询当前选定测量报告的列数 |
| :COLUMNHEADER? | 查询指定索引处的列名称 |
| :DATA? | 查询指定行和列索引处的数据 |
| :REFRESH | 重新计算测量报告 |
| :ROWCOUNT? | 查询当前选定测量报告的行数 |
| :SAVE | 将当前选定的测量报告保存到文件 |
| *LA:TIMINGREPORT | 用于获取和保存时序报告数据的命令集 |
| :COLUMNCOUNT? | 查询当前选定时序报告的列数 |
| :COLUMNHEADER? | 查询指定索引处的列名称 |
| :DATA? | 查询指定行和列索引处的数据 |
| :LOADRESULT? | 加载时序报告过程的结果 |
| :LOADSETTING | 加载时序报告的设置 |
| :ROWCOUNT? | 查询当前选定时序报告的行数 |
| :SAVE | 将当前选定的时序报告保存到文件 |
| :TESTRESULT? | 测试过程的结果 |
| *LA:TRANSITIONREPORT | 用于获取和保存跳变报告数据的命令集 |
| :COLUMNCOUNT? | 查询当前选定跳变报告的列数 |
| :COLUMNHEADER? | 查询指定索引处的列名称 |
| :CURSORPOSITION? | 查询光标的位置 |
| :DATA? | 查询指定行和列索引处的数据 |
| :ROWCOUNT? | 查询当前选定跳变报告的行数 |
| :SAVE | 将当前选定的跳变报告保存到文件 |
| :SELECT | 选择跳变报告列表的索引 |


Copyright © 2025 Acute Technology Inc.

## 4.4.3 命令定义

***LA:CAPTURE:START**

启动波形采集。

- **语法**:

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

- **参数**:

  无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_CAPTURE_ALREADY_RUNNING | 表示采集已在进行中。 |

- **viRead() 返回值**:

  无

- **示例**:

```
// 启动逻辑分析仪采集
viWrite("*LA:CAPTURE:START");
```

***LA:CAPTURE:STOP**

停止波形采集。

- **语法**:

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

- **参数**: 无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_CAPTURE_NOT_RUNNING | 表示应用程序未启动采集。 |

- **viRead() 返回值**:

  无

- **示例**:

```
// 停止逻辑分析仪采集
viWrite("*LA:CAPTURE:STOP");
```


Copyright © 2025 Acute Technology Inc.

***LA:CAPTURE:PROGRESS?**

读取采集进度。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  表示当前采集进度的值，范围从 0.00 到 1.00。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:CAPTURE:PROGRESS?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("采集进度 = %f\n", atof(buffer));
```


Copyright © 2025 Acute Technology Inc.

***LA:CAPTURE:TRIGGERED?**

读取采集触发状态。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  `1`: 触发事件已发生。

  `0`: 触发事件未发生。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:CAPTURE:PROGRESS?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("采集已触发 = %d\n", atoi(buffer));
```



Copyright © 2025 Acute Technology Inc.

***LA:CAPTURE:STATUS?**

读取采集状态。


Copyright © 2025 Acute Technology Inc.

- **语法:**

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

- **参数:**

  无

- **viRead() 返回值:**

  表示当前采集状态的字符串。

| 状态 | 描述 |
| --- | --- |
| Data Transferring | 数据正在传输中。 |
| Data Processing | 数据正在处理中。 |
| Capture Finished | 逻辑分析仪已完成数据采集。 |
| Analyzing | 逻辑分析仪正在分析数据。 |
| Analysis Finished | 逻辑分析仪已完成数据分析。 |
| Pre-Trigger | 逻辑分析仪进入预触发状态。 |
| Wait For Trigger | 逻辑分析仪正在等待触发条件。 |
| Post-Trigger | 逻辑分析仪进入后触发状态。 |
| Wait For DSO Capture | 逻辑分析仪正在等待 DSO 采集。 |
| Capturing | 逻辑分析仪正在采集数据。 |
| Saving Report | 逻辑分析仪正在保存报告。 |
| Report Save Finished | 逻辑分析仪已完成报告保存。 |
| File Load Finished | 逻辑分析仪已完成文件加载。 |

- **示例:**

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:CAPTURE:STATUS?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}
```


Copyright © 2025 Acute Technology Inc.

```
// 显示结果
printf("采集状态 = %s\n", buffer);
```

#### *LA:CURSOR:DELETE

删除选定的光标。

- **语法**:

```
// 删除所有光标
*LA:CURSOR:DELETE -ALL
```

```
// 删除指定光标
*LA:CURSOR:DELETE <cursorN>
```

- **参数**:

  *-ALL*: 删除所有光标。

  *<cursorN>*: 删除指定光标。

- **viRead() 返回值**: 无
- **示例**:

```
// 删除所有光标
viWrite("*LA:CURSOR:DELETE -ALL");
```

```
// 删除光标 B
viWrite("*LA:CURSOR:DELETE CursorB");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:CURSOR:SEARCH:SET:BYEDGE

将光标移动/搜索条件设置为边沿跳变。

- **语法**:

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

- **参数**:

  *<label>*: 设置目标标签，例如 CH-04

  *<edgeCount>*: 设置条件边沿计数。

  *<edgeType>*: 设置条件边沿类型。

- **viRead() 返回值**: 无
- **示例**:

```
// 将光标移动/搜索条件设置为 CH-05 上升沿 x8
viWrite("*LA:CURSOR:SEARCH:SET:BYEDGE CH-05 8 RISE");
```

```
// 将光标移动/搜索条件设置为 CH-15 下降沿 x10
viWrite("*LA:CURSOR:SEARCH:SET:BYEDGE CH-15 10 FALL");
```

#### *LA:CURSOR:SEARCH:SET:BYTIME

将光标移动/搜索条件设置为时间周期。

- **语法**:

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

- **参数**:

  *<time>*: 设置时间条件，例如 10ns、50ms

- **viRead() 返回值**: 无
- **示例**:

```
// 将光标移动/搜索条件设置为 50.5us
viWrite("*LA:CURSOR:SEARCH:SET:BYTIME 50.5us");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:CURSOR:SEARCH:SET:BYVALUEMATCH

将光标移动/搜索条件设置为值或文本匹配。

- **语法**:

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

- **参数**:

  *<label>*: 设置目标标签，例如 CH-04

  *<value>*: 设置值/文本匹配条件，例如 05h 或 "Data Read"

- **viRead() 返回值**: 无
- **示例**:

```
// 将光标移动/搜索条件设置为值 50h（十六进制）
viWrite("*LA:CURSOR:SEARCH:SET:BYVALUEMATCH 50h");
```

```
// 将光标移动/搜索条件设置为文本匹配 "Data Read"
viWrite("*LA:CURSOR:SEARCH:SET:BYVALUEMATCH \"Data Read\"");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH

将光标移动/搜索条件设置为脉冲宽度。

- **语法**:

```
*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>
```

- **参数**:

  *<label>*: 设置目标标签，例如 CH-04

  *<pulseType>*: 设置脉冲类型为高电平、低电平或任意。

  *<conditionType>*:

```
- G: 大于
- L: 小于
- GE: 大于或等于
- LE: 小于或等于
- E: 等于
- NE: 不等于
- INRANGE: 在范围内，需要指定两个时间条件。
- OUTRANGE: 在范围外，需要指定两个时间条件。
```

  *<time>*: 设置时间条件，例如 10ns、50ms

- **viRead() 返回值**: 无

- **示例**:

```
// 将光标移动/搜索条件设置为 CH-03 正脉冲宽度为 10 us
viWrite("*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH CH-03 HIGH L 10us");
```

```
// 将光标移动/搜索条件设置为 CH-13 任意脉冲宽度在 1 ms 和 10 ms 之间
viWrite("*LA:CURSOR:SEARCH:SET:SEARCHPULSEWIDTH CH-13 ALL INRANGE 1ms 10ms");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:CURSOR:SEARCH:START

开始光标移动/搜索。

- **语法**:

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

- **参数**:

  *<direction>*: 设置搜索方向为左或右。

  *<cursor>*: 设置目标光标，例如 CursorA、CursorB、... 、CursorZ

  *MULTIPLE <count>*: 设置移动多个光标，需要指定额外的光标移动计数，例如 5。

- **viRead() 返回值**: 无
- **示例**:

```
// 从光标 C 开始，向右搜索并放置光标 C 到 E。
viWrite("*LA:CURSOR:SEARCH:START RIGHT CursorC MULTIPLE 3");
```

#### *LA:WAVEFORM:SCALE

设置波形的水平时间刻度。

- **语法**:

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

- **参数**:

  *time*: 水平波形时间刻度，单位：ps、ns、us、ms、s。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将波形时间刻度设置为 10 us
viWrite("*LA:WAVEFORM:SCALE 10us");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:WAVEFORM:MOVETO

将波形中心移动到指定光标位置。

- **语法**:

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

- **参数**:

  *cursor*: 光标名称，例如 CursorA、CursorB、... 、CursorZ

- **viRead() 返回值**: 无
- **示例**:

```
// 将波形的中心点移动到光标 C 的位置
viWrite("*LA:WAVEFORM:MOVETO CURSORC");
```

#### *LA:WAVEFORM:STORE

存储当前波形位置或时间刻度。

- **语法**:

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

- **参数**:

  *-SCALE*: 存储当前波形时间刻度。

  *-POSITION*: 存储当前波形位置。

- **viRead() 返回值**: 无
- **示例**:

```
// 存储当前波形位置
viWrite("*LA:WAVEFORM:STORE -POSITION");
```

```
// 存储当前波形时间刻度
viWrite("*LA:WAVEFORM:STORE -SCALE");
```

```
// 同时存储当前波形位置和时间刻度
viWrite("*LA:WAVEFORM:STORE -POSITION -SCALE");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:WAVEFORM:RESTORE

恢复之前的波形位置或时间刻度。

- **语法**:

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

- **参数**:

  *-SCALE*: 恢复之前的波形时间刻度。

  *-POSITION*: 恢复之前的波形位置。

- **viRead() 返回值**: 无
- **示例**:

```
// 恢复之前的波形位置
viWrite("*LA:WAVEFORM:RESTORE -POSITION");
```

```
// 恢复之前的波形时间刻度
viWrite("*LA:WAVEFORM:RESTORE -SCALE");
```

```
// 同时恢复之前的波形位置和时间刻度
viWrite("*LA:WAVEFORM:RESTORE -POSITION -SCALE");
```

#### *LA:WINDOWSIZE:REPORTSIZE

设置报告区域大小，同时也会改变波形区域大小。

- **语法**:

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

- **参数**:

  *SIZE*: 指定报告区域大小，范围从 0 到 100（单位：百分比）。

- **viRead() 返回值**: 无
- **示例**:

```
// 将报告区域大小设置为屏幕的 50%
viWrite("*LA:WINDOWSIZE:REPORTSIZE 50");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:WINDOWSIZE:WAVEFORMSIZE

设置波形区域大小，同时也会改变报告区域大小。

- **语法**:

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

- **参数**:

  *SIZE*: 指定波形区域大小，范围从 0 到 100（单位：百分比）。

- **viRead() 返回值**: 无
- **示例**:

```
// 将波形区域大小设置为屏幕的 50%
viWrite("*LA:WINDOWSIZE:WAVEFORMSIZE 50");
```


Copyright © 2025 Acute Technology Inc.

***LA:DECODEREPORT:COLUMNCOUNT?**

获取当前解码报告的列数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  当前解码报告的列数

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:DECODEREPORT:COLUMNCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("解码报告列数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

***LA:DECODEREPORT:COUNT?**

获取解码报告数量。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  当前解码报告数量

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:DECODEREPORT:NAME?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("解码报告数量 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *LA:DECODEREPORT:DATA?

获取选定索引处的解码报告数据。

- **语法**:

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

- **参数**:

  *row*: 解码报告行索引。

  *column*: 解码报告列索引。

- **viRead() 返回值**:

  选定索引处的解码报告数据。

- **示例**:

```c
// 使用 viWrite() 发送命令
if (!viWrite("*LA:DECODEREPORT:DATA? 2 1")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

printf("位置 (2, 1) 处的解码报告数据 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

***LA:DECODEREPORT:NAME?**

获取当前解码报告名称。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  当前解码报告名称

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:DECODEREPORT:NAME?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("解码报告名称 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

***LA:DECODEREPORT:ROWCOUNT?**

获取当前解码报告的行数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  当前解码报告的行数

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:DECODEREPORT:ROWCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("解码报告行数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *LA:DECODEREPORT:SAVE

保存解码报告。

- **语法**:

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

- **参数**:

  *filename*: 目标文件名。

  *-BEGIN <row> (可选)*: 输入报告保存的起始行，默认从报告开始处开始。

  *-END <row> (可选)*: 输入报告保存的结束行，默认到报告结束处结束。

  *-NOTS (可选)*: 报告保存功能在保存报告的第一列存储时间戳信息，添加此参数指定不在保存的报告中存储时间戳信息。

  *-USESEC (可选)*: 报告保存功能将以纳秒为单位存储时间戳信息，添加此参数指定使用秒来存储时间戳信息。

  *-NOHEADER (可选)*: 指定不在保存的报告中包含头信息。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将报告保存到 MyDecodeReport.csv
viWrite("*LA:DECODEREPORT:SAVE MyDecodeReport.csv");
```

> **Note**
>
> 请确保您有权限写入目标文件目录。

#### *LA:DECODEREPORT:SELECT

选择解码报告。

- **语法**:

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

- **参数**:

  *index*: 解码报告索引。

- **viRead() 返回值**:

  无

- **示例**:

```
// 选择第一个解码报告
viWrite("*LA:DECODEREPORT:SELECT 0");
```



Copyright © 2025 Acute Technology Inc.

#### *LA:MEASUREMENTREPORT:COLUMNCOUNT?

获取测量报告的列数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回当前测量报告的列数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:MEASUREMENTREPORT:COLUMNCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("测量报告列数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *LA:MEASUREMENTREPORT:COLUMNHEADER?

获取指定列索引处的测量报告列标题名称。

- **语法**:

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

- **参数**:

  *column*: 指定列索引。

- **viRead() 返回值**:

  返回指定列索引处的当前测量报告列标题名称。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:MEASUREMENTREPORT:COLUMNHEADER? 1")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("索引 1 处的测量报告列标题 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

***LA:MEASUREMENTREPORT:DATA?**

获取指定行和列索引处的测量报告数据。

- **语法**:

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

- **参数**:

  *row*: 指定行索引。

  *column*: 指定列索引。

- **viRead() 返回值**:

  指定行和列索引处的测量报告数据。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:MEASUREMENTREPORT:DATA? 1 2")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("位置 [1, 2] 处的测量报告数据 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

#### *LA:MEASUREMENTREPORT:REFRESH

重新计算测量报告。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  无

- **示例**:

```
// 重新计算测量报告
viWrite("*LA:MEASUREMENTREPORT:REFRESH");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:MEASUREMENTREPORT:ROWCOUNT?

获取测量报告的行数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回当前测量报告的行数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:MEASUREMENTREPORT:ROWCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("测量报告行数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *LA:MEASUREMENTREPORT:SAVE

将测量报告保存到文件。

- **语法**:

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

- **参数**:

  *filename*: 目标文件名。

  *-BEGIN <row> (可选)*: 输入报告保存的起始行，默认从报告开始处开始。

  *-END <row> (可选)*: 输入报告保存的结束行，默认到报告结束处结束。

  *-NOTS (可选)*: 报告保存功能在保存报告的第一列存储时间戳信息，添加此参数指定不在保存的报告中存储时间戳信息。

  *-USESEC (可选)*: 报告保存功能将以纳秒为单位存储时间戳信息，添加此参数指定使用秒来存储时间戳信息。

  *-NOHEADER (可选)*: 指定不在保存的报告中包含头信息。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将测量报告保存为 .csv 文件到 MyMeasurementReport.csv
viWrite("*LA:MEASUREMENTREPORT:SAVE MyMeasurementReport.csv");
```

```
// 将测量报告保存为 .html 文件到 MyMeasurementReport.html
viWrite("*LA:MEASUREMENTREPORT:SAVE MyMeasurementReport.html");
```

> **Note**
>
> 请确保您有权限写入目标文件目录。



Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:COLUMNCOUNT?

获取时序报告的列数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  当前时序报告的列数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:COLUMNCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("时序报告列数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:COLUMNHEADER?

获取时序报告列标题。

- **语法**:

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

- **参数**:

  *column*: 时序报告列索引。

- **viRead() 返回值**:

  当前时序报告列标题。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:COLUMNHEADER? 1")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("索引 1 处的时序报告列标题 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:DATA?

获取指定行和列索引处的时序报告数据。

- **语法**:

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

- **参数**:

  *row*: 时序报告行索引。

  *column*: 时序报告列索引。

- **viRead() 返回值**:

  指定行和列索引处的时序报告数据。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:DATA? 1 2")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("位置 [1, 2] 处的时序报告数据 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:LOADRESULT?

加载时序报告结果。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回 *LA:TIMINGREPORT:LOADSETTING 的处理结果， 如果没有错误则返回 "OK"，否则返回错误信息。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:LOADRESULT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("时序报告加载结果 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:LOADSETTING

加载时序报告设置。

- **语法**:

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

- **参数**:

  *filename*: 时序报告设置文件名。

- **viRead() 返回值**:

  无

- **示例**:

```
// 加载时序报告设置
viWrite("*LA:TIMINGREPORT:LOADSETTING MyTimingSettings.csv");
```

> **Note**
>
> 请确保您有权限写入目标文件目录。



Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:ROWCOUNT?

获取时序报告的行数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回当前时序报告的行数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:ROWCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("时序报告行数 = %d\n", atoi(buffer));
```

> **Note**
>
> 请确保您有权限写入目标文件目录。



Copyright © 2025 Acute Technology Inc.

#### *LA:TIMINGREPORT:SAVE

将时序报告保存到文件。

- **语法**:

```
// 将时序报告保存为 .csv 文件
*LA:TIMINGREPORT:SAVE -CSV <filename>
```

```
// 将时序报告保存为 .txt 文件
*LA:TIMINGREPORT:SAVE -TXT <filename>
```

```
// 将时序报告保存为 .html 文件
*LA:TIMINGREPORT:SAVE -HTML <filename> [-TITLE <title>] [-USERINPUT <userInput>]
```

- **参数**:

  *filename*: 目标文件名。

  *-CSV*: 将报告保存为 .csv 文件。

  *-TXT*: 将报告保存为 .txt 文件。

  *-HTML*: 将报告保存为 .html 文件。

  *-TITLE <title> (可选)*: 指定 HTML 报告标题。

  *-USERINPUT <userInput> (可选)*: 指定用户输入信息。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将时序报告保存为 .csv 文件到 MyTimingReport.csv
viWrite("*LA:TIMINGREPORT:SAVE –CSV MyTimingReport.csv");
```

```
// 将时序报告保存为 .html 文件到 MyTimingReport.html 并带有标题
viWrite("*LA:TIMINGREPORT:SAVE –HTML MyTimingReport.html –TITLE My Example Title");
```

> **Note**
>
> 请确保您有权限写入目标文件目录。



Copyright © 2025 Acute Technology Inc.

***LA:TIMINGREPORT:TESTRESULT?**

获取时序报告的整体通过/失败结果。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  如果所有项目测试都通过则返回 `Pass`，否则返回 `Fail`。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TIMINGREPORT:TESTRESULT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("时序报告测试结果 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

***LA:TRANSITIONREPORT:COLUMNCOUNT?**

获取跳变报告的列数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回当前跳变报告的列数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TRANSITIONREPORT:COLUMNCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("跳变报告列数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

***LA:TRANSITIONREPORT:COLUMNHEADER?**

获取指定列索引处的跳变报告列标题名称。

- **语法**:

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

- **参数**:

  *<column>*: 指定列索引。

- **viRead() 返回值**:

  返回指定列索引处的当前跳变报告列标题名称。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TRANSITIONREPORT:COLUMNHEADER? 1")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("位置 [1] 处的跳变报告列标题名称 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TRANSITIONREPORT:DATA?

获取指定行和列索引处的跳变报告数据。

- **语法**:

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

- **参数**:

  *row*: 指定行索引。

  *column*: 指定列索引。

- **viRead() 返回值**:

  指定行和列索引处的跳变报告数据。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TRANSITIONREPORT:DATA? 1 2")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("位置 [1, 2] 处的跳变报告数据 = %s\n", buffer);
```



Copyright © 2025 Acute Technology Inc.

***LA:TRANSITIONREPORT:ROWCOUNT?**

获取跳变报告的行数。

- **语法**:

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

- **参数**:

  无

- **viRead() 返回值**:

  返回当前跳变报告的行数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TRANSITIONREPORT:ROWCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("跳变报告行数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

### *LA:TRANSITIONREPORT:SAVE

将跳变报告保存到文件。

- **语法**:

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

- **参数**:

  *filename*: 目标文件名。

  *-BEGIN <row> (可选)*: 输入报告保存的起始行，默认从报告开始处开始。

  *-END <row> (可选)*: 输入报告保存的结束行，默认到报告结束处结束。

  *-NOTS (可选)*: 报告保存功能在保存报告的第一列存储时间戳信息，添加此参数指定不在保存的报告中存储时间戳信息。

  *-USESEC (可选)*: 报告保存功能将以纳秒为单位存储时间戳信息，添加此参数指定使用秒来存储时间戳信息。

  *-NOHEADER (可选)*: 指定不在保存的报告中包含头信息。

- **viRead() 返回值**:

  无

- **示例**:

```
// 将跳变报告保存为 .csv 文件到 MyTransitionReport.csv
viWrite("*LA:TRANSITIONREPORT:SAVE MyTransitionReport.csv");
```

```
// 将跳变报告保存为 .html 文件到 MyTransitionReport.html
viWrite("*LA:TRANSITIONREPORT:SAVE MyTransitionReport.html");
```

> **Note**
>
> 请确保您有权限写入目标文件目录。



Copyright © 2025 Acute Technology Inc.

#### *LA:TRANSITIONREPORT:SELECT

选择标签并刷新跳变报告。

- **语法**:

```
// 选择所有标签
*LA:TRANSITIONREPORT:SELECT -ALL
```

```
// 选择特定标签
*LA:TRANSITIONREPORT:SELECT <label1> <label2> ... <labelN>
```

- **参数**:

*-ALL*: 选择所有标签。

*<labelN>*: 选择特定标签。

- **viRead() 返回值**:

无

- **示例**:

```
// 选择标签并刷新跳变报告
viWrite("*LA:TRANSITIONREPORT:SELECT");
```

```
// 选择所有标签以创建跳变报告
viWrite("*LA:TRANSITIONREPORT:SELECT -ALL");
```

```
// 选择标签 CH-00、Bus[7:0] 和 CH50 以创建跳变报告
viWrite("*LA:TRANSITIONREPORT:SELECT CH-00 Bus[7:0] CH50");
```


Copyright © 2025 Acute Technology Inc.

#### *LA:TRANSITIONREPORT:CURSORPOSITION?

获取指定光标所在的跳变报告行号。

- **语法**:

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

- **参数**:

  *<cursor>*: 指定光标名称，例如 CursorA、CursorB、... 、CursorZ

- **viRead() 返回值**:

  返回指定光标所在的当前跳变报告行号。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*LA:TRANSITIONREPORT:CURSORPOSITION? CursorA")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("跳变报告光标位置 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

# 4.5 Protocol Analyzer Mode Subsystem

## 4.5.1 什么是协议分析仪模式？

**协议分析仪**模式允许您捕获和分析串行协议数据。 它为各种串行协议（如 I2C、SPI、UART 等）提供专门的解码和分析功能。 协议分析仪模式通过将原始信号解码为特定协议的事务并以易于理解的格式呈现，帮助工程师调试和验证串行通信。

主要功能包括：

- 实时解码协议信号
- 详细的事务视图和统计信息
- 数据导出和报告

## 4.5.2 命令格式

本节描述协议分析仪子系统的命令格式。 协议分析仪子系统中的每个命令都以 ***PA:** 开头。

下表是协议分析仪子系统的命令树。

| 命令 | 描述 |
| --- | --- |
| *PA:CAPTURE | 用于捕获和分析协议数据的命令集 |
| :START | 启动波形捕获 |
| :STOP | 停止波形捕获 |
| *PA:REPORT | 用于查询和保存报告数据的命令集 |
| :ROWCOUNT? | 获取报告行数 |
| :COLUMNCOUNT? | 获取报告列数 |
| :SAVE | 保存报告到文件 |
| :DATA? | 获取选定索引处的报告数据 |


Copyright © 2025 Acute Technology Inc.

## 4.5.3 命令定义

***PA:CAPTURE:START**

启动协议分析仪捕获。

- **语法**:

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

- **参数**:

  无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_CAPTURE_ALREADY_RUNNING | 表示采集已在进行中。 |

- **viRead() 返回值**:

  无

- **示例**:

```
// 启动协议分析仪采集
viWrite("*PA:CAPTURE:START");
```

***PA:CAPTURE:STOP**

停止协议分析仪捕获。

- **语法**:

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

- **参数**:

  无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_CAPTURE_NOT_RUNNING | 表示应用程序未启动采集。 |

- **查询值**:

  无

- **示例**:

```
// 停止协议分析仪采集
viWrite("*PA:CAPTURE:STOP");
```


Copyright © 2025 Acute Technology Inc.

***PA:REPORT:ROWCOUNT?**

获取报告的行数。

- **语法**:

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

- **参数**:

  无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |

- **viRead() 返回值**:

  报告的行数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*PA:REPORT:ROWCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("行数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

#### *PA:REPORT:COLUMNCOUNT?

获取报告的列数。

- **语法**:

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

- **参数**:

  无

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |

- **viRead() 返回值**:

  报告的列数。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*PA:REPORT:COLUMNCOUNT?")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("列数 = %d\n", atoi(buffer));
```


Copyright © 2025 Acute Technology Inc.

***PA:REPORT:SAVE**

使用指定类型将报告保存到指定文件名的文件中。

- **语法**:

```
*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]
```

- **参数**:

  *filename*: 目标文件名。

  *-SAVEALL (可选)*: 将整个报告保存到一个文件中。

  *-ADVRPT (可选)*: 包含高级报告数据。

  *-SPLITDUR (可选)*: 将报告时间戳和持续时间数据分成两列。

  *-SAVESPLIT <row> (可选)*: 根据行数将报告分割成多个文件。

  *-SAVERANGE (可选)*: 保存选定行和列范围内的报告。

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_INPUT_FILE_DIR_INVALID | 表示输入文件名无效。 |

- **viRead() 返回值**:

  无

- **示例**:

```
// 将整个报告保存到一个文件中
viWrite("\*PA:REPORT:SAVE C:/Temp/AllReport.csv -SAVEALL");
```

```
// 将整个报告及高级报告数据保存到一个文件中
viWrite("\*PA:REPORT:SAVE C:/Temp/AdvReport.csv -SAVEALL -ADVRPT");
```

```
// 根据行数 = 2000 将报告分割成多个文件
viWrite("\*PA:REPORT:SAVE C:/Temp/LimitRpt.csv -SAVESPLIT 2000");
```

```
// 保存选定行和列范围内的报告
viWrite("\*PA:REPORT:SAVE C:/Temp/RangedRpt.csv -SAVERANGE 200 2000 1 5");
```



Copyright © 2025 Acute Technology Inc.

#### *PA:REPORT:DATA?

从选定索引获取报告数据。

- **语法**:

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

- **参数**:

  *row*: 行索引。

  *column*: 列索引。

- **状态码**:

| 状态码 | 描述 |
| --- | --- |
| AQVI_NO_ERROR | 命令执行成功。 |
| AQVI_ROW_COL_INDEX_INVALID | 表示列索引无效。 |

- **viRead() 返回值**:

  选定索引处的报告数据。

- **示例**:

```
// 使用 viWrite() 发送命令
if (!viWrite("*PA:REPORT:DATA? 105 2")) {
    return; // viWrite 失败
}

// 等待软件处理命令
while (viGetCommandResult() == PROC_PENDING) {
    ProcessWaitAction();
}

// 检查命令是否执行成功
if (viGetCommandResult() != PROC_OK_WITH_RETURN) {
    // 没有返回数据或处理命令失败
    return;
}

// 使用 viRead() 从应用程序缓冲区获取响应
char buffer[32] = {0};
if (!viRead(buffer, sizeof(buffer))) {
    return; // viRead 失败
}

// 显示结果
printf("索引 [105, 2] 处的报告数据 = %s\n", buffer);
```


Copyright © 2025 Acute Technology Inc.

Copyright © 2025 Acute Technology Inc.

# 5. Error Codes

## 5.1 状态码表

| 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.