The Journal of Open Source Software

# SwiftVISA: Controlling Instrumentation with a Swift-based Implementation of the VISA Communication Protocol

**Connor Barnes**[1], **Luke Henke**[1], **Lorena Henke**[2], **Ivan Krukov**[1], **and Owen Hildreth**[1¶]

**1** Colorado School of Mines **2** Consolidated Analysis Center, Incorporated ¶ Corresponding author

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:**

**Submitted:** 17 March 2022
**Published:** unpublished



## Summary

The Virtual Instrument Software Architecture (VISA)Contributors (2021) is a simple Application Programming Interface (API) implementing the Standard Commands for Programmable Instruments (SCPI)(Foundation, 2021) to control test and measurement instrumentation from a computer. Over the years, VISA has been wrapped or ported to other languages in an effort to make it easier to write programs to control instruments and analyze their data in real time. This paper introduces `SwiftVISASwift`,(Barnes, 2021d) a pure Swift-based VISA implementation for use on both x86 and ARM-based processors. `SwiftVISASwift` allows scientists and engineers to write full-featured, native iOS and macOS applications while leveraging standardized SCPI commands to control their instrumentations. Unlike other VISA implementations, such as `pyVISA`, `SwiftVISASwift` provides native access to standard iOS and macOS API's and works well with asynchronous programming methodologies. This paper details the design and basic use of SwiftVISA.

## Statement of Need

The VISA framework is a key backbone for many research instruments. It's SCPI syntax is versatile, easy to learn, and adaptable to low and high throughput data streams. It's original C implementation makes it easy to wrap a pre-compiled framework (such as `NI-VISA`) in different languages. `pyVISA` is one of the most widely known and used implementation of this strategy.(pyVISA, 2014) `pyVISA` is highly portable between operating systems and python itself is relatively easy to learn. However, wrapping the VISA framework comes with the limitations associated with any wrapper. For example, the VISA framework hasn't been compiled for ARM processors and `pyVISA` (and other wrappers) are limited to x86 processors. Additionally, pre-compiled frameworks aren't as portable as packaged-based distribution methods. `pyVISApy` has worked to address some of these issues by re-implementing VISA entirely in Python.(PyVISA-py, 2014) However, `pyVISApy` still doesn't have access to the Operating System's APIs.

Inspired by `pyVISA` and `pyVISApy`, we have developed `SwiftVISASwift` as a Swift-based VISA package that doesn't require a pre-compiled VISA framework. This package allows developers to write full featured macOS and iOS applications to control their instruments. Since `SwiftVISASwift` is delivered as a package instead of a framework, it can be compiled against a wide range of hardware platforms and supports x86 and ARM processors and can be used in Universal Binaries. This manuscript will summarize `SwiftVISASwift` and example some simple code examples.

The Journal of Open Source Software

## SwiftVISASwift Overview

SwiftVISASwift has evolved over the years from SwiftVISA, a pyVISA inspired wrapper around the pre-compiled NI-VISA C-framework, to an intermidate backend service combined with higher level APIs implemented as a Swift Package, to finally a pure Swift implementation of the VISA famework.

The 'SwiftVISASwift' package is broken into four sub-implementations to provide developers with the widest design freedoms. These are:

- CoreSwiftVISA(Barnes, 2021a)
- SwiftVISASwift(Barnes, 2021d)
- NISwiftVISA(Barnes, 2021b)
- SwiftVISA (deprecated)(Barnes, 2019)

The figure below shows how these implementations work together and provide options to developers based upon their needs. SwiftVISA and NISwiftVISA still depend on the pre-compiled NI-VISA framework and provides access to GPIB, USB, and TCI/IP communication. However, since they ultimately rely on NI-VISA, they are limited to x86 systems and, at the time of this writing, to macOS 11 and below (National Instruments hasn't yet released NI-VISA that works on macOS 12 yet. The SwiftVISASwift package uses the underlying CoreSwiftVISA package to provide a pure Swift implementation that works on x86 and ARM processors and is compatible with macOS 12. However, only TCI/IP communcation has been implemented for SwiftVISASwift due to limitations applied to the kext currently used to communicate with USB devices.

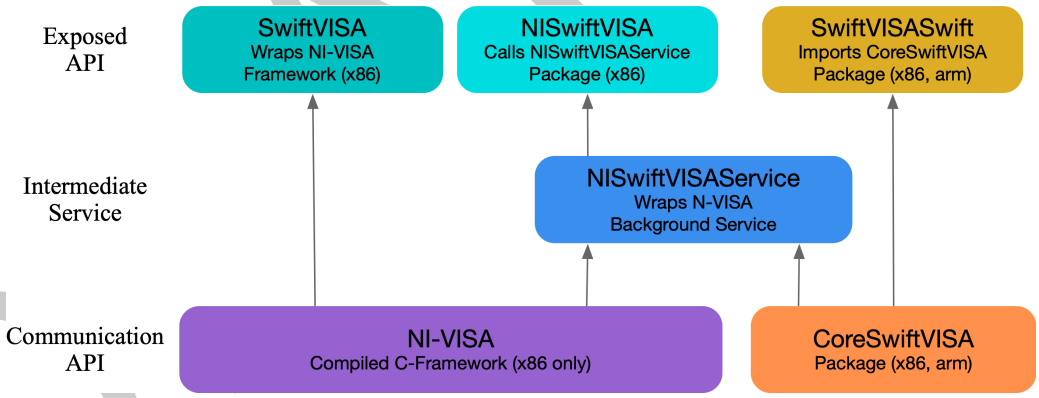



**Figure 1:** Schematic showing how the implementions integrate together

CoreSwiftVISA is a low-level package that provides a base, underlying implementation of SwiftVISA, excluding the communication implementation portion (i.e. TCI/IP, USB, etc.). This includes defining base types and protocols. CoreSwiftVISA isn't directly used, instead, it is used by higher-level packages, such as SwiftVISASwift and NISwiftVISA. Breaking the core components out into a separate package makes it simpler to abstract away implementation details. This can be used to create custom backends for SCPI-compliant instruments or other types of instruments.

SwiftVISASwift uses CoreSwiftVISA types and protocols and implements communication over TCI/IP instruments. This is a pure Swift, native backend that does not require installation of the VISA or NI-VISA frameworks. SwiftVISASwift is currently limited to communicating with TCI/IP instruments because communication on iOS and macOS devices now requires signed drivers. While it is possible to use older USB drivers, macOS has started restricting kexts and breaking out the implementation into those that did and didn't require signed drivers was considered more future-proof.

NISwiftVISA allows for communicating over USB instruments. This implementation does use the NI-VISA C backend and requires that users have NI-VISA 20.0 or later installed. Additionally, users must install NISwiftVISAService.(Barnes, 2021c) The NISwiftVISAService is a process that runs in the background that allows the NISwiftVISA framework to communicate with NI-VISA's C framework. This service is necessary because NI only distributes pre-compiled binaries of its NI-VISA framework and, at we wrote NISwiftVISA, Swift Packages did not support dependencies on pre-compiled binaries within a package. While Swift 5.3 introduced dependencies for binary dependencies, they were limited to XCFrameworks. NI-VISA is not an XCFramework. To circumvent this requirement, the NISwiftVISA uses interprocess communication that calls the NISwiftVISAService, which handles all communicates to the instruments. NISwiftVISA works well on macOS systems, but is not compatible with iOS systems. Additionally, unless the NI-VISA framework is updated to a universal framework, it will run under ARM systems (e.g. Apple Silicon) through Apple's Rosetta 2 dynamic binary translator. Just as Apple's Rosetta 1 translator was eventually Obsolete and unavailable, we can expect Rosetta 2 to also be phased out eventually.

SwiftVISA was our original implementation and is simply a wrapper, like pyVISA, around the NI-VISA framework. This requires that NI-VISA framework is installed and is limited to macOS devices. SwiftVISA is considered deprecated at this point and we do not expect to provide any future updates. Even though it is deprecated, SwiftVISA still functions and we have tested it up to macOS 11.1 (if SIPS is disabled). It does not work on ARM processors and doesn't work on macOS 12 at this time.

## Features

SwiftVISASwift allows direct communication with SCPI-compliant devices in an easy, and accessible manner. The SwiftVISASwift implementation requires no external frameworks for TCI/IP devices while the NISwiftVISA implementation allows for communication with USB and GPIB (along with TCI/IP) devices while using the NISWiftVISAService background process to bypass the need for importing the NI-VISA framework directly into your package or application. Additionally, SwiftVISASwift is compatible with Swift 5.5's new built-in support for writing asynchronous and parallel code in a structured way. This includes concurrancy features such as async-await, actor, Task. As a result, it is easy to define instrument controllers as Actors and ensure that sending commands and collecting data from the instrument doesn't tie up the main thread or GUI.

The following example demonstrates some basic functionalities. A complete example demonstrating a typical workflow using the SwiftVISASwift package with a TCI/IP compatible device.

SwiftVISASwift is first imported as a package through Swift Package Manager. To use SwiftVISASwift in your project, include the following dependency in your Package.swift file.

```
dependencies: [.package(url: "https://github.com/SwiftVISA/SwiftVISASwift.git", .upToNex
```

To create a connection to an instrument over TCP/IP, pass the network details to InstrumentManager.shared.instrumentAt(address:port:). Since this operation can throw an error if it fails you can either use a do-catch statement to handle the error or have the calling function throw the error up your chain.

```
do {
    // Pass the IPv4 or IPv6 address of the instrument to "address" and the insturment's
    let instrument = try InstrumentManager.shared.instrumentAt(address: "10.0.0.1", port
} catch {
    // Could not connect to insturment
    // Handle the error
}
```

To write to the instrument, call `write(_:)` on the instrument:

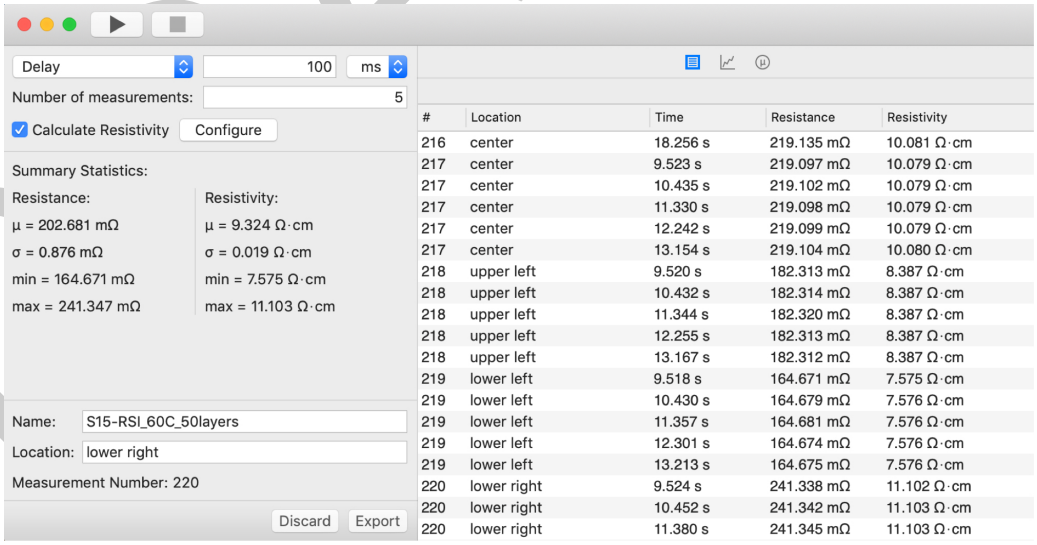
```
do {
    // Pass the command as a string.
    try instrument.write("OUTPUT ON")
} catch {
    // Could not write to insturment
    // Handle the error
}
```

To read from the instrument, call `read()` on the instrument:

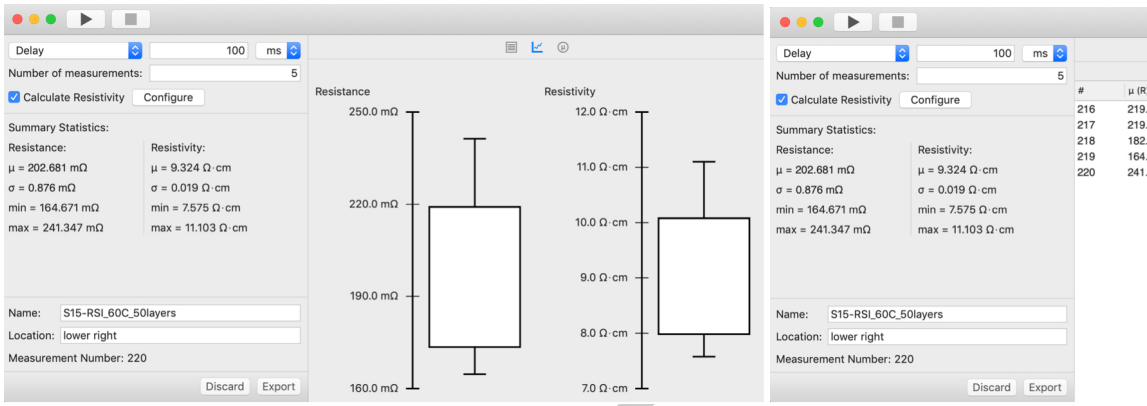
```
do {
    try instrument.write("VOLTAGE?")
    let voltage = try instrument.read() // read() will return a String
} catch {
    // Could not read from (or write to) insturment
    // Handle the error
}
```

## Example Applications

Internally, we've been using `SwiftVISA` and `SwiftVISASwift` for a few years now to control various laboratory hardware. Resistivity Utility is an example application written with `SwiftVISASwift` to control an Agilent Nanovolt meter connected to a custom 4-point probe station. This simple application lets us document our measurements, apply the appropriate geoemtry corrections for resistivy calculations, and export both the raw and summarized data to csv files.(Barnes, 2020)



The Journal of Open Source Software




## Acknowledgements

We would like to acknowledge the financial support we received from the National Science Foundation (Award #: CAREER 401756).

## References

Barnes, C. (2019). *SwiftVISA*. https://github.com/SwiftVISA/SwiftVISA

Barnes, C. (2021a). *CoreSwiftVISA*. https://github.com/SwiftVISA/CoreSwiftVISA

Barnes, C. (2021b). *NISwiftVISA*. https://github.com/SwiftVISA/NISwiftVISA

Barnes, C. (2021c). *NISwiftVISAService*. https://github.com/SwiftVISA/NISwiftVISAService

Barnes, C. (2021d). *SwiftVISASwift*. https://github.com/SwiftVISA/SwiftVISASwift

Barnes, C. (2020). https://github.com/seeaya/Resistivity-Utility

Contributors, W. (2021). *Wikipedia page on VISA*. https://en.wikipedia.org/wiki/Virtual_instrument_software_architecture

Foundation, I. (2021). *SCPI overview*. https://www.ivifoundation.org/scpi/

Instruments, N. (2021). *NI-VISA download*. https://www.ni.com/en-us/support/downloads/drivers/download.ni-visa.html#409839

pyVISA. (2014). *pyVISA*. https://pyvisa.readthedocs.io/en/latest/

PyVISA-py. (2014). *PyVISA-py*. https://github.com/pyvisa/pyvisa-py