Computer Physics Communications 234 (2019) 216–222

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Lab::Measurement—A portable and extensible framework for controlling lab equipment and conducting measurements☆

S. Reinhardt [a], C. Butschkow [a], S. Geissler [a], A. Dirnaichner [a,1], F. Olbrich [a,2], C.E. Lane [b], D. Schröer [c], A.K. Hüttel [a,*]

[a] Institute for Experimental and Applied Physics, University of Regensburg, 93040 Regensburg, Germany
[b] Department of Physics, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104, USA
[c] Center for NanoScience and Department für Physik, Ludwig-Maximilians-Universität, Geschwister-Scholl-Platz 1, 80539 München, Germany

ARTICLE INFO

Article history:
Received 10 April 2018
Received in revised form 9 July 2018
Accepted 31 July 2018
Available online 10 August 2018

Keywords:
Measurement control
GPIB
USB T&M
Ethernet
VXI-11
VISA
SCPI
Perl

ABSTRACT

Lab::Measurement is a framework for test and measurement automatization using Perl 5. While primarily developed with applications in mesoscopic physics in mind, it is widely adaptable. Internally, a layer model is implemented. Communication protocols such as IEEE 488 [1], USB Test & Measurement [2], or, e.g., VXI-11 [3] are addressed by the connection layer. The wide range of supported connection backends enables unique cross-platform portability. At the instrument layer, objects correspond to equipment connected to the measurement PC (e.g., voltage sources, magnet power supplies, multimeters, etc.). The high-level sweep layer automates the creation of measurement loops, with simultaneous plotting and data logging. An extensive unit testing framework is used to verify functionality even without connected equipment. Lab::Measurement is distributed as free and open source software.

Program summary

Program Title: Lab::Measurement 3.660
Program Files doi: http://dx.doi.org/10.17632/d8rgrdc7tz.1
Program Homepage: https://www.labmeasurement.de
Licensing provisions: GNU GPL v2[3]
Programming language: Perl 5
Nature of problem: Flexible, lightweight, and operating system independent control of laboratory equipment connected by diverse means such as IEEE 488 [1], USB [2], or VXI-11 [3]. This includes running measurements with nested measurement loops where a data plot is continuously updated, as well as background processes for logging and control.
Solution method: Object-oriented layer model based on Moose [4], abstracting the hardware access as well as the command sets of the addressed instruments. A high-level interface allows simple creation of measurement loops, live plotting via GnuPlot [5], and data logging into customizable folder structures.

[1] F. M. Hess, D. Penkler, et al., LinuxGPIB. Support package for GPIB (IEEE 488) hardware, containing kernel driver modules and a C user-space library with language bindings. http://linux-gpib.sourceforge.net/
[2] USB Implementers Forum, Inc., Universal Serial Bus Test and Measurement Class Specification (USBTMC), revision 1.0 (2003). http://www.usb.org/developers/docs/devclass_docs/
[3] VXIbus Consortium, VMEbus Extensions for Instrumentation VXIbus TCP/IP Instrument Protocol Specification VXI-11 (1995). http://www.vxibus.org/files/VXI_Specs/VXI-11.zip
[4] Moose—A postmodern object system for Perl 5. http://moose.iinteractive.com
[5] E. A. Merritt, et al., Gnuplot. An Interactive Plotting Program. http://www.gnuplot.info/

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: andreas.huettel@ur.de (A.K. Hüttel).

[1] Current address: Potsdam Institute for Climate Impact Research, Telegraphenberg A 31, 14473 Potsdam, Germany.
[2] Current address: TechBase Sensorik-ApplikationsZentrum, Franz-Mayer-Str. 1, 93053 Regensburg, Germany.
[3] The precise license terms are more permissive. Lab::Measurement is distributed under the same licensing conditions as Perl 5 itself, a frequent choice in

https://doi.org/10.1016/j.cpc.2018.07.024

1. Introduction

Experimental physics frequently relies on complex instrumentation. While high-level equipment often has built-in support for automation, and while more and more ready-made solutions for complex workflows exist, the nature of experimental work lies in the development of new workflows and the implementation of new ideas. A freely programmable measurement control system is the obvious solution.

One of the most widely used applications of this type is National Instruments LabVIEW [1], where programming essentially means drawing the flowchart of your application. While this is highly flexible and supported by many hardware vendors, more complex use cases can easily lead to flowcharts that are difficult to understand and to maintain. A second widely used commercial application implementing instrument control is MathWorks MATLAB [2], where this ties into its numerical data evaluation and graphical user interface (GUI) capabilities.

`Lab::Measurement` provides a lightweight, open source alternative based on Perl 5, a well-known and well-established script programming language with a strong emphasis on text manipulation. `Lab::Measurement` has been successfully used for quantum transport measurements on GaAs/AlGaAs gate-defined quantum dots, see, e.g., [3–5], as well as carbon nanotube devices [6–8] and other mesoscopic systems [9,10].

As other open source measurement packages, see e.g. [11,12], `Lab::Measurement` features a modular structure which facilitates adding new hardware drivers. Here we highlight the unique features of `Lab::Measurement`. This includes in particular its high-level, descriptive "sweep" interface, where data and metadata preservation as well as real-time plotting is intrinsically provided. Notable is further the automated creation of unit tests via connection logging, and the use of traits [13] to exploit the modular structure of SCPI [14] and thereby minimize code duplication.

2. Implementation and architecture

`Lab::Measurement` is implemented as an object-oriented layer structure of Perl modules, as displayed schematically in Fig. 1. The figure does not show the full set of provided modules, but demonstrates a small selection of hardware.

The uppermost layer depicted in the figure, corresponding to the *hardware driver*, is not part of the `Lab::Measurement` Perl module distribution. Several Perl binding modules such as `Lab::VISA` (comparable to the Python PyVISA module [15]), `Lab::VXI11`, `Lab::Zhinst`, and `USB::TMC` are separately provided by the `Lab::Measurement` authors via the Comprehensive Perl Archive Network (CPAN), as bindings to third-party libraries. Conversely, e.g., the LinuxGPIB package [16] provides a Perl interface to its library and Linux kernel modules on its own.

The *connection layer* provides a unified API to these hardware drivers, making it possible to address devices, send commands, and receive responses. It handles access to many common automation protocols, such as GPIB, USB-TMC, VXI-11, raw TCP Sockets, or for example the Zurich Instruments LabOne API.

An overview of the currently supported connection types is given in Table 1.

The *instrument layer* represents connected test and measurement devices. The generic instrument class `Lab::Moose::Instrument` only provides means to send commands and receive responses; derived classes encapsulate the model-specific command sets. On this level, direct device programming can take place.

the Perl ecosystem. This means that it can be used and distributed according to the terms of either the GNU General Public License (version 1 or any later version) or the Artistic License; the choice of license is up to the user.

```
package Lab::Moose::Instrument::SR830;

use Moose;
use Lab::Moose::Instrument qw/validated_getter
                              validated_setter/;
use Lab::Moose::Instrument::Cache;

extends 'Lab::Moose::Instrument';
with qw(
  Lab::Moose::Instrument::Common
);

# the initialization
sub BUILD {
  my $self = shift;
  $self->clear(); $self->cls();
}

# reference signal frequency: declare cache
cache frq => ( getter => 'get_frq' );

# reference signal: get frequency
sub get_frq {
  my ( $self, %args ) = validated_getter( \@_ );
  return $self->cached_frq(
    $self->query( command => 'FREQ?', %args ) );
}

# reference signal: set frequency
sub set_frq {
  my ( $self, $value, %args )= validated_setter(
    \@_, value => { isa => 'Num' } );
  $self->write(
              command => "FREQ $value", %args );
  $self->cached_frq($value);
}

# [...]

# measured value, get r and phi in one call
sub get_rphi {
  my ( $self, %args ) = validated_getter( \@_ );
  my $retval = $self->query(
                 command => "SNAP?3,4", %args );
  my ( $r, $phi ) = split( ',', $retval );
  chomp $phi;
  return { r => $r, phi => $phi };
}

# [...]
```

Listing 1: Commented excerpt from the driver module for the Stanford Research SR830 lock-in amplifier; shown are the header and initialization, the functions for getting and setting the reference voltage frequency, and a function for reading out radius and phase of the measured signal.

The *sweep layer* finally provides abstraction of sweeping instrument parameters and performing multi-dimensional measurements, with live plotting and data logging. This facilitates the creation of complex measurement tasks for non-programmers without creation of loops or conditional control flow statements.

3. Extending and programming the instrument layer

3.1. Minimal example of an instrument driver

Listing 1 shows a commented excerpt of an instrument driver module, based on the Stanford Research SR830 lock-in amplifier module distributed with `Lab::Measurement`.

Low level functionality in drivers is frequently implemented via Moose roles. These are an implementation of the concept of traits, which enable horizontal code reuse more efficiently than single inheritance, multiple inheritance, or mixins [13,17]. In our case, roles are used to share functions between different instrument drivers. This significantly eases implementation of new drivers compared to other open-source measurement frameworks, e.g., [11,12]. The minimal driver of Listing 1 only consumes a single role, `Lab::Moose::Instrument::Common`, providing, e.g., the