Open source Instrumenation Platform for Measurement and Testing Systems

Undergraduate thesis

David Šibrava

1.5.2017

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Cybernetics

Project supervisor: doc. Ing. Josef Vedral, CSc.

Study programme: Cybernetics and Robotics

Specialisation: Robotics

(This page is intentionally left blank.)

Table of content

(This page is intentionally left blank.)

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: David Šibrava

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Open Source Instrumentation Platform for Measurement and Testing Systems

Guidelines:

Design and create an instrumentation platform for measuring and testing system, allowing the collection, preprocessing and digitalization of voltage and current signals. The scope of the work will include the design of a user interface, allowing users to display calculations made using the data collected and display results of measurments and the related calculations, including the posibility to export the data into table processors.
Flexibility of the user interface should be one of the main design goals.

Bibliography/Sources:

[1] Horowitz, P.: The Art of Electronics. Cambridge University Press, 2015.
[2] Haasz, V., Roztočil, J., Novák, J.: Číslicové měřicí systémy. Vydavatelství ČVUT 2000.
[3] Vedral, J.,Fischer, J.: Elektronické obvody pro měřicí techniku. Vydavatelství ČVUT 2004.

Bachelor Project Supervisor: doc. Ing. Josef Vedral, CSc.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 9, 2017

Abstract

This document discusses the work I have done on my long-term instrumentation platform project as a part of my Bachelor thesis. In future, the instrumentation platform will consist of modules, which connect to a PC and software, which can be used to collect data from those modules, perform calculations using this data and also to handle basic automation. The work done for my Bachelor thesis mostly consists of writing said software and writing a bytecode compiler and virtual machine for the language, usable for basic automation tasks.

However, I have also created a basic proof-of-concept module and have defined the protocol usable to communicate between the PC and the said module.

Keywords:

Instrumentation platform, measurement, user interface, GUI, module, compiler, CLUC,

Java, Vim, Ranger

Abstrakt

Tento dokument pojednává o části mého dlouhodobého projektu instrumentační platformy, kterou jsem zpracoval jako svou bakalářskou práci. V budoucnu se mnou vyrobená platforma bude sestávat z modulů, připojitelných k PC a softwaru, který půjde použít na sběr dat z těchto modulů, pro provedení výpočtů nad takto sesbíranými daty a také pro provedení základní automatizace.

Pro účely této bakalářské práce jsem se rozhodl soustředit se převážně na napsání výše zmíněného softwaru a na napsání kompilátoru do bytekódu a virtuálního stroje pro jazyk, který by byl použitelný pro základní automatizaci.

Nicméně jsem též vytvořil jednoduchý modul pro demonstrační účely a následně definoval

komunikační protokol, kterým se tento může dorozumět s PC.

Klíčová slova:

Instrumentační platforma, měření, uživatelské rozhraní, GUI, modul, kompilátor, CLUC,

Java, Vim, Ranger

List of attachments:

Videos:

1) GUI_autogeneration https://youtu.be/QtyAM0vie4c
2) mappings_macros https://youtu.be/QEDLpuugU14
3) gui_creation https://youtu.be/Ru1cBu1woRY
4) chart_evalAt https://youtu.be/s4vnSgv7Ilc
5) stats_processing https://youtu.be/6va4i3vuDLw
6) selection_subsystem https://youtu.be/c8odhMq4ELY
7) GUI_editing https://youtu.be/QtyAM0vie4c

Java projects:

1) GUI project
2) Compiler project

Further documentation:

1) GUI project Javadoc
2) language.docx (CLUC language specification)

Other:

1) Further development.docx

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ……..

signature: ……..

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne……..

Podpis autora práce: ……..

Scope of my work and tools used

This short section is intended to summarize what tools I used and to what extend is the end result my work, and to what extent I used already available tools and other resources.

I have made most of this project myself. In fact, retrospectively, I might say I overdid it in terms of what all I made and looking back, I could have used existing projects, instead of resorting to making almost everything from scratch. However, at the time, doing everything myself really seemed like a good idea.

Some of the GUI concepts and ideas (and the general philosophy) are taken from the Vim and Vi text editors [1]; others are taken from the Ranger file manager [2]. I explicitly mention the sources of inspiration where applicable.

This being said, I have only used those programs as my source of inspiration and haven't actually reused any code of those or similar projects.

I have used the Netbeans IDE [3] for all the Java programming I did. I have also heavily utilized the jVi plugin [4] for said IDE, which makes it behave more like my favorite text editor, Vim. [1]

I have used the Java classes and tools available with the standard Java SE Development Kit 8 installation. [5] Additionally, I have also used the JavaFX 8.0 [6]software Platform from Oracle and the RXTX Library by Trent Jarvi. [7]

JavaFX is a GUI library (Library for creating Graphical User Interfaces) with many interesting and powerful features; [6] however, I have mostly only used its key processing routines and then the Canvas element [8], which allows various shapes and texts to be drawn freely onto the application window and I've introduced my own GUI elements and concepts.

I have used the "Working with Canvas" [9] tutorial by Scott Hommel as a starting point.

The original reasoning behind this decision was that I later wanted to port my application to a different, lower computing power platform, which might not support JavaFX, but would still support some means of drawing basic shapes on the screen. Not relying on JavaFX heavily would make such transition easier.

The RXTX Library is a library providing a native interface to serial ports in Java. [7] I have used its low level features, such as opening a serial port for communication and sending bytes over it.

I have built the proof-of-concept module using the Arduino Duemillanove hardware [10] and the Arduino IDE [11]as a programming tool; No additional software was used for the development for this module and no libraries, other than the ones that are included by default in the fresh Arduino IDE installation were used.

The CLUC language, which is a programming language used for the automation of my platform was also built from scratch by me; the work included coming up with the formal language definition, writing a parser, bytecode compiler and 2 virtual machines, one written in Java and one in C. No automated compiler/parser generators, such as Bison [12] or Yacc [13], were used.

I have used the Vim text editor [1] and the Markdown [14] language to write this document and the documents attached. I have then used Pandoc [15] to convert them to a .docx format, where I proceeded to edit them in Microsoft Word 2010. [16]

The screenshots not from cited sources are taken by me.

The camcorder icon used next to video links was drawn by me in Adobe After effects CS6.

The videos demonstrating the functionality of my projects are directed, recorded and edited by me.

Introduction

I have always been fascinated by electronics. Since it has always been my hobby, I often deal with equipment used for creation, testing and debugging of electronic circuits, such as multimeters or power supplies.

However, it has always bothered me that for decades, those instruments haven't virtually changed at all. While they can fit certain use cases really well, I find them to be vastly inadequate for others. Most measuring and instrumentation equipment accessible to a hobbyist like me has certain downsides:

Many multimeters, for instance, do not offer any PC connectivity [17], or they do, but the customer needs to pay extra [18], [19]; the situation is similar with power supplies. Most multimeters don't allow their users to perform calculations on the measured data in real time. [17] Unless we are dealing with professional grade equipment, it's difficult to get multiple instruments to cooperate with each other, for instance, to ensure they have both taken their samples at similar times. Without putting a great effort in the task, it's often impossible to perform even basic automation, such as have the voltmeter alert us with a sound when the measured voltage gets outside a specified area.

There are many other problems that I have found in current "consumer grade" test equipment. In section Why an instrumentation platform?, I have discussed the aforementioned issues more in-depth, explaining why I believe they need to be addressed.

It goes without saying that attempts have been made, by people other than myself, to address those issues. Indeed, various complex systems, which are usually referred to as instrumentation platforms, exist.

Those systems usually consist of modules, which can measure various electrical or non-electrical quantities. It's then possible to record the data from those modules and process it in PC. However, even if it's fair to say that such platforms address many issues I had with "traditional" measuring equipment, most of them they either don't match my expectations, are way outside my reach financially, or in most cases both. Namely, such platforms are usually not open source and "hackable." It's not possible to (easily) connect, say, the ever-popular Arduino development board to them and use it as a sensor. It's not possible to easily develop your own sensor and use the provided software to collect data from it.

They usually do not allow any sort of automatization. Their functionality can seldom be scripted.[1] In other words, it's not possible to execute actions (that could, for instance, affect the outputs of some modules) whenever a condition (such as an input module reading appearing in a certain range) is fulfilled.

Other platforms do offer this functionality, but in a very limited manner.

Some of such platforms are intended for data collection only and do not even have any output modules.

Rigidity of the user interface of the controlling program is one of my other concerns.

Ideally, it should be possible to quickly reorganize the different GUI elements, responsible for displaying the measured data and allowing the user to control outputs of the platform. It should be possible to change both the behavior and appearance of such GUI elements.

It is often the case, that while it is often possible to reorganize and edit the user interface of the controlling program to one’s own liking, it is not an easy task. This doesn't come as a surprise, since most of the Instrumentation platforms I've looked into are either intended for educational purposes, or for scientific research. In such use cases, it is seldom necessary to change the user interface; moreover, the person responsible for editing such interface is different from the person using it.

Admittedly, the layout of the different viewers (such as displays or charts) and controls, ideal for conducting an experiment in classroom can be put together by a teacher once and then used by scholars for years to come.

However, an electronic hobbyist should be able to quickly adjust the controls to his own liking, as quickly as possible.

---

[1] These statements are provided without a citation, as they are based on the brief research, findings of which are covered in the Similar projects section of this document.

They may be working on multiple projects simultaneously; each requires their own measuring tasks and thus poses a slightly different requirement for the ideal layout of the resulting user interface.

Since it would be no use to discuss the downsides and upsides of instrumentation platforms in general any further, as vast differences prevent such generalization, I have instead conducted a quick research on some of such platforms.

I have made remarks about upsides and downsides of the projects I found, that are most similar to my idea of an instrumentation platform in the section "Similar projects."

To summarize the results of this research, I have come up with a conclusion, that no instrumentation platform project fully matches my criteria.

Naturally, this means I have to create my own instrumentation platform from scratch.

This is a long-term goal, which can be broken down into 3 easy steps:

1) Create modules, able to measure various quantities, such as voltage or current, or output (actuators).
2) Come up with some means of communication between such modules and a PC
3) Design software, able to display data measured by the modules, analyze it statistically, display results of calculations using such data in real time, export measured or computed quantities into spreadsheet editors, provide some means of automation (event driven actions) and more

For my bachelor thesis, I have mostly focused on step 3). This being said, I have also designed the communication protocol and even have successfully created a proof-of-concept hardware, able to measure voltage levels and send them to the software for processing.

Having given it a throughout thought, I have concluded my requirements for my instrumentation platform to be following:

- The platform should consist of a PC-based software and modules that connect to this PC. The modules can each have inputs (they can measure quantities) or outputs (actuators). Each module can have multiple inputs and multiple outputs. User interface is the means to control the outputs and read the inputs.

- The modules should carry information about basic user interface they want to provide. When they get connected to the PC, such basic user interface, allowing the user to read out values measured by the module and control its outputs should appear. That is, the module must be able to command the PC to add GUI elements and change any of their properties.

- Aforementioned user interface needs to be divided into tabs, similar to tabs known from internet browsers. Once created, it should be easy to edit it by removing or moving the GUI elements, even between individual tabs. It must be possible to change functional and aesthetic properties of said elements, such as position, display size or

colors. It should also be possible to create additional GUI elements and place them on the tabs.

- It has to be possible to calculate equations, using the measured values, and display results of such equations in the GUI elements, created by the user. Such equations can contain numbers, properties of the GUI elements (such as the displayed value), constants, function calls and more. Function can either be defined algorithmically, or as a series of points, connected by a line. Results of such equations need to be displayed in real time or further processed.
- In addition to calculating equations, using the current values as input, it would be practical to be able to do statistical analysis. For instance, the user may need to calculate a floating average, minimum, maximum, divergence, sum and other statistical indicators from the measured and computed values.
- It would definitely be really nice to plot measured or computed value in chart in real time. Plotting multiple values simultaneously is a logical requirement, as well as the option to record multiple data lines in a single chart. Obviously, once the data is collected this way, it should be possible to export it into spreadsheet editors for further processing. Support for plotting histogram would be very nice, too!
- It should be possible to execute actions whenever conditions are met; for instance, to automatically adjust outputs of the output modules whenever the measured inputs meet certain criteria.

The work I have done fulfills all the requirements listed above. I have created a simple proof-of-concept module which measures voltages and output them, which communicates with a PC in real time.

Then I have written software, which handles all the tasks described. It consists of 2 main parts: The user interface and CLUC interpreter.

CLUC stands for C-like Language for User Code and is a simple C-inspired language designed by me, which handles all the automatization, equation parsing and more.

The language is precompiled and then interpreted from a bytecode.

Both the compiler and the matching virtual machine have been written by me from scratch.

They are covered in a separate document.[2]

What is an instrumentation platform?

Instrumentation platform is a platform for measurement, testing and control. It typically consists of a rack ("chassis" or main module), to which many different modules are

[2] See the List of attachments: section.

connected. Sometimes, the main unit is actually a typical PC, as is the case in my instrumentation platform project.

The individual modules provide different sourcing and measurement functions - they can measure voltage, current, frequency, or generate a waveform, provide a constant current/voltage or switch loads. The main unit or a PC provides some means of controlling those modules, as well as insures the synchronization between them.

Why an instrumentation platform?

In this chapter, I'd like to briefly cover some of the benefits a centralized instrumentation platform (one main module and purpose-built secondary modules connected to it) has in comparison with the alternative, which is having many different dedicated standalone instruments.

There are many reasons to build an open source instrumentation platform. Here are some of the benefits of this approach, compared to the alternative, which would be getting separate instruments.

- cost

  Although instrumentation platforms are not really well known for their low price (from the perspective of a hobbyist), many different functions can be offloaded to the main module, allowing the other modules to be much simpler, which reduces the overall cost significantly. Aforementioned functions can include, but are not limited to:

  - User interface (Displaying the measured quantities and/or controlling the outputs)
  - part of the processing power
  - power supply and power supply related circuitry

  Furthermore, by handling the user interface on the main unit, the manufacturers and designers of the individual modules are not required to provide their own user interface implementations, which makes the hardware design easier, whilst potentially also reducing its size, and allows an easier use of off-the-shelf, readily available project boxes for the module housing, which furthermore reduces the price. For instance, a power supply module would normally need to have a display and knobs in order to be controllable. This complicates the design and adds manufacturing steps (such as making the cutouts for the knobs and more.) When the user interaction is handled externally, the design complexity and thus the cost of the module itself is reduced.

The fact, that a lot of functionality is offloaded to the main unit simplifies not only the hardware, but also the software running in the module.

- extensibility

  Platform-module approach solution is much more futureproof than using multiple devices. When the user needs a feature, which is not supported by their current equipment, they typically need to replace it. Not only does this come with a significant

cost, but it also requires a non-negligible amount of time, spent on choosing the replacement, learning to operate it and getting it to work with other devices.

Having one central platform and being able to get specific single-purpose modules for it allows users to get new features when they need them. This also means they are not required to have features they don't need.

Furthermore, in this particular project, its open-source nature will allow more experienced users to design and make their own modules.

- ability to correlate measurements

  Having multiple modules take measurements with one device having access to them allows those measurements to be correlated. This makes it possible to measure property/property relations, such as temperature vs. power output, voltage vs. current, frequency vs. voltage and more. This is either impossible, or difficult with standard, hobbyist accessible devices. Additionally, this property may even enable the system to watch phenomena, that would otherwise be impossible or beyond impractical to measure. Examples include U/I characteristics of a diode, or a measurement of a dissipated power using a voltmeter and an ammeter.

- high end features on low end devices

  Sometimes, user does not desire an extreme accuracy or very low ranges of measurement. However, he might still want to log the data. However, advanced devices that allow datalogging would be pointlessly accurate and therefore would impose a substantial, pointless financial burden. Platform approach allows giving even low end devices high end features.

For the sake of objectivity, downsides of this approach are listed below

- relying on a central unit

  If the central unit breaks down for whatever reason, all the devices that rely on it are affected. Therefore, special care must be taken to ensure the possibility of this happening is minimal. Measures include full galvanic isolation between the mane unit and the individual module, as well as mains earth isolation.

- lack of novelty

  Getting a new device is always rewarding for a hobbyist or any person who is buying it for the sake of their pleasure. In the case of one centralized platform, getting a new module doesn't come with as much "novelty factor" as, say, getting a new dedicated power supply.

Goals of the project

The primary goal of the project is to create an open source platform that could then be used by electronics hobbyists, people doing repair jobs and diagnostics, people involved in (mostly small scale) hardware development and those in the need for cheap and flexible platform for both measurement and automation.

It goes without saying that, given how flexible the design is, the resulting product can be used for virtually any purpose, ranging from education to home automation, or even as a generic IOT platform. This being said, the target demography mainly consists of electronics engineers and hobbyists, which is also reflected by the form of the user interfaces (to be covered).

Part of this goal shall be completed as this bachelor thesis and the rest as the Diploma thesis.

For the time being, I wanted to develop a PC-side software, which can communicate with some simple, proof-of-concept hardware, display the data generated by such hardware, process it, do statistical analysis on it, etc.

In some distant future, I'd like to use a small PC, such as Orange-Pi, use a small display with it, put it in an enclosure and really make the whole thing standalone without the need for a "big" pc.

For my bachelor thesis, I have developed a very simple module, able to measure multiple voltage levels simultaneously. This module also has some output capabilities. It only serves as a proof of concept - I needed a device that sends data to the PC for further processing. I have mostly focused on the software part of development for my Bachelor thesis, with the intention on focusing on the hardware part during the Diploma thesis.

Open-source-ness

I believe that stating some project is formally open source is virtually pointless, as long as it cannot *practically* be treated as such. It is often the case that the code of such a project is so cluttered and poorly documented that the time that one would need to spend in order to decrypt the allegedly open source code surpasses the time that would be needed for a full rewrite. Therefore, one of my goals is to keep the documentation up to date and make sure it covers all the important parts. For this reason, I wrote a separate document describing the inner workings of the CLUC language compiler/interpreter and wrote a Javadoc (code documentation) for the software.

Real-time data analysis

What even some of the high end instruments lack is the ability to flexibly analyze the data in real time, as opposed to analyzing them in post. For instance, when we measure voltage and current, we might want to multiply those two in order to get power where applicable. In some cases, we want to perform this multiplication in real time. Similarly, we might have a PTC thermistor and we want to get the temperature. Most instruments do not allow their

users to simply plug in the resistance-temperature calculation in order to get the temperature display in real time. One of the goals is therefore to provide flexible means of data analysis using real-time operations, expressed by complex math expressions, on the measured data.

Data acquisition

It should be possible to collect the data from various modules and store them for future offline analysis, using tools specifically designed for the task, such as table processors or Matlab and Matlab compatible/inspired software. Furthermore, it should be possible to display charts and histograms in real-time as the data is getting collected.

User interface development goals

PC side GUI

Time and time again have I found overall rather good products, which I find very lacking on the user interface side. This is a great shame, as the lack of effort put into designing such a user interface greatly reduces the overall utility of the resulting system. One such example would be the PC-side GUI for the Korad3005D power supply that I own.

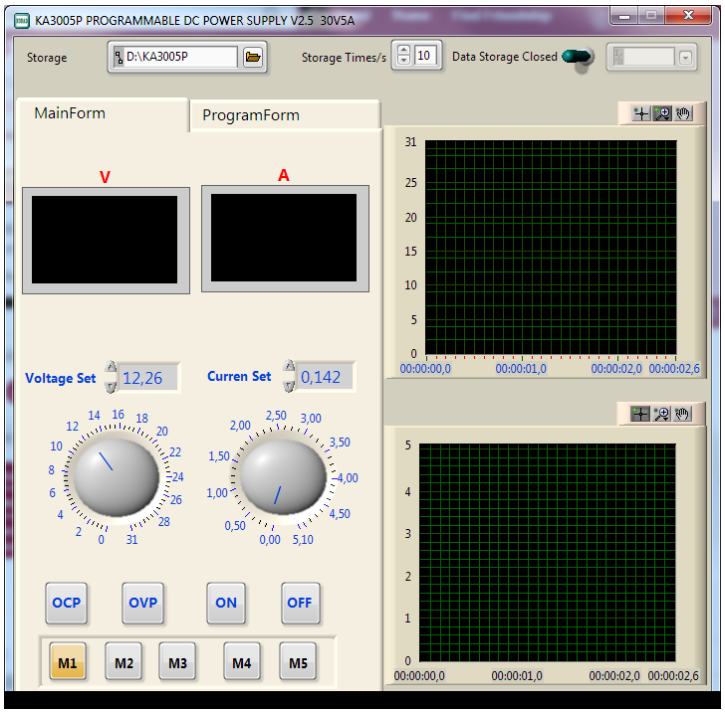


Image 1 - GUI for the KORAD 3005D power supply

Although it's beyond the scope of this document to cover why the said user interface is bad, let me give one example of its "features" that I find rather unacceptable:

Sometimes, when the user dials the "voltage set" dial to zero, it underflows to the maximum of 31 volts (!); this value then actually gets set on the power supply output (!!!). The behavior of the current limit is similar. Furthermore, scrolling the mouse wheel while hovering the mouse over the dial increments or decrements the value by adding one to the least significant digit of the currently displayed voltage. That is, 8.98 jumps to 8.99 in one mouse wheel click, from which it then jumps to 9 - so far so good. However, the next mouse wheel click gives you the value of 10V (!) and the next even gives you 20 (!!!) V.

There are often no easy ways to solve this problem with the lacking user interface. Its behavior can rarely be adjusted due to its inflexibility and even the communication protocols are sometimes proprietary (not really the case with the aforementioned power supply, which uses SCPI). In order to be satisfied, the user would need to reverse engineer the communication protocol used by his device and then design his own GUI around it, which is a time and labor intensive process. Therefore, I find it absolutely crucial to develop a user interface system that will not only be extremely powerful and give the user freedom to do anything he wants to, but also can be extremely customizable. In my experience, the PC programs with best (i.e. fastest to use and most flexible) user interfaces are usually heavily keyboard oriented.

Moreover, I'd like to, at some point in the future, get rid of the PC dependency and have a standalone main module. This module will be built around a small pc platform, such as Orange PI and will run a slightly modified version of the software I now run on a PC.

Such main module will have its own set of physical controls - knobs, buttons, etc. I believe that if my software can, if desired, be controlled from keyboard exclusively, will make it easier to conform the existing user interface to the aforementioned set of physical controls.

If I were to pick one single program with the best user interface, it would probably be the very popular Vi(m) text editor. We do not need to go far to have a proof that its user interface is a true masterpiece - all it takes to realize is to look at the list of programs that have purposefully, with more or less success tried to replicate its UI to some extent. [20]

It includes programs that I use and am happy with, such as Vimperator, Ranger, jVi or i3.

The Vi/Vim user interface paradigm has repeatedly proven itself to be very powerful and flexible; it is also something many engineers are already familiar with; therefore, one of the goals of my project is to make use of the said paradigm to the largest extent possible.

In addition, the user interface should be scriptable. That is, it should be possible to write a set of actions that get executed once a predicate is met. This way it should be possible to create user programs and implement any sort of desired functionality.

Connectivity

At the moment, child modules are connected to the master module (PC) over a serial port.

For further development, I'll need to either change the connection type or use a galvanically isolated serial port.

Hardware design

As mentioned, I have primarily been focusing on the software side of the problem.

Therefore, I have only created hardware necessary to demonstrate the power of the software developed. I have quickly put together a board, which acts as a so-called "shield" for Arduino Duemillanove/Uno. This shield consists of:

Inputs:

- 1 PTC thermistor
- 1 photoresistor
- 1 button
- 1 potentiometer

Outputs:

- 3 LEDs (Red, Green and Blue)

I am using the built-in ADC of the Atmega328 processor, which the Arduino board is based around, to read out the values of analog inputs and send them to the PC.

I can also control the intensity of the LEDs present on my simple shield, using pulse-width modulation.

This being said, I have conducted some brief research regarding the hardware I will want to use, so I can briefly summarize the results:

At around $15 retail price, Orange PI is a very inexpensive PC platform. [21] It shouldn't take much effort to use in place of a regular desktop PC; this would make the platform truly standalone.

ESP8266 and ESP32 development boards are also really inexpensive and have Wi-Fi support. Modules built around them could measure wirelessly and send data; that would definitely be really interesting.

One other part that caught my eye is the MCP3901 analog frontend. [22] It's inexpensive (~$2-3), contains 2 separate ADCs, programmable gain amplifiers, low-drift voltage reference, differential inputs, it offers extremely high resolution (16 bits oversampled to 24), is TTL compatible and can be controlled over a simple SPI interface.

Similar projects

Before I decided to create my own instrumentation platform, I researched the market briefly, to see if something affordable, that would be matching my criteria, hasn't already been created.

I wanted a platform, which would:

- be open source and hackable, that is, users would be able to create their own modules for it and contribute to the development
- allow computing equations in real time, using the measured data
- plot results of such equations in real time and allow users to export them into spreadsheet editors
- allow some degree of automation; that is, allow its users to setup actions, that shall, for instance, change the outputs of some modules whenever a condition is met
- would have a flexible user interface, that can quickly be adjusted to match the measurement requirements
- be inexpensive

I have picked several projects that I believe are sufficiently similar to mine and have written down notes about them below.

National instruments LabView

Introduction

One of the most popular environments for automated testing, control and data acquisition is the LabView IDE. [23] As ambitious as my project and its creator are, it would be a mistake to try competing against a set of tools created by large groups of experts in the field over the course of 30+ years [24].

Admittedly, the power of Labview isn't easy to match. However, as to 11.11.2016, the price of Labview was over $1500. [25] This is usually far outside the reach of an electronics hobbyist. It would be a valid argument to note that most people will pirate an illegal copy of LabView - however, the special hardware (such as the NIs RIO or CompactRIO platforms) that works with Labview is usually also really pricey and is therefore impossible to acquire. In fact, while NI does sell some devices for measurement and data acquisition at relatively low costs, none of their modular platforms are affordable. While it IS indeed possible to use SOME hardware not made by National Instruments with LabView and it is a common practice, it may require a non-trivial amount of setting up and might yield synchronization problems. Additionally, the GUI of LabView is heavily mouse oriented and while being okay for an occasional use, the lack of keyboard support will slow down the workflow, especially when repetitive tasks are carried out. LabView and related product also seem to have some limited support for Android and IOS phones, which is definitely their benefit.

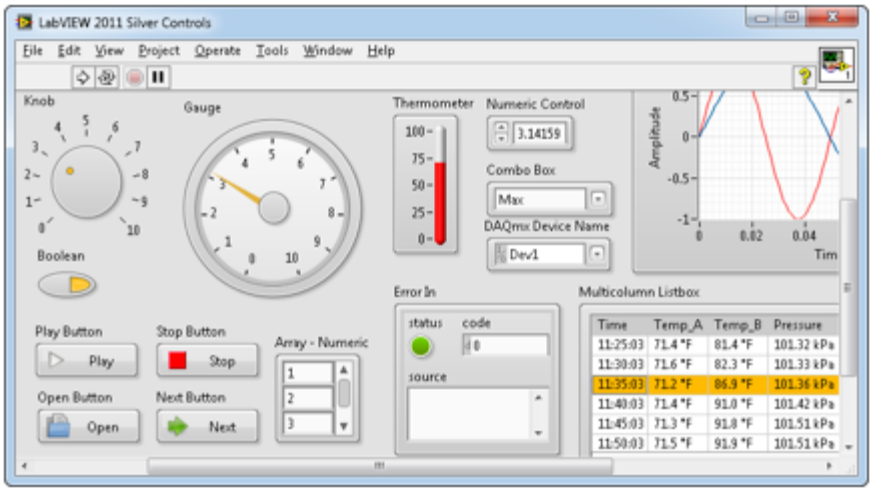


Image 2 - LabView GUI [26]

Remark: I have recently (just now, while searching web for pictures to put in this document, 1.5.2017) learned about the fact that LabView does have a so called "Home Bundle" version, targeted at makers and hobbyists. [27] This version is for noncommercial purposes only, and currently retails for $50. I find that rather unfortunate, given how much work I've put into my project; I find it rather confusing that I haven't learned about this earlier. However, I still do believe that my project is superior at least in some aspects. I am not sure what those are, since I do not have the resources to conduct further research regarding LabView.

This being said, the final product, once everything is completed, should be a modular, fully standalone piece of test gear, which is very different from LabView.

ISES

Introduction

ISES is a Czech product developed on the Faculty of mathematics and physics, Charles University. [28] It is a rather popular and old electronic modular system, aimed at the education of physics, chemistry and biology. It is primarily designed to be used for measurement.

Image 3- the ISES platform [28], [29]

User interface

The software supports online analysis of the data collected and its GUI is very adjustable. However, it cannot be easily adjusted on the fly. It is really intended to be created by the teacher once and then used over and over without changes to it.

ISES also has some support for web browser interfaces, making it possible to take measurements remotely from smartphones or tablets.

Automation

I haven't managed to find any information on whether or not, or to what extent, can the platform be used for automation. I don't suppose it's possible in a simple manner though.

Output and control

The only output modules listed on the official page [30] are:

- booster module

This module probably consists of a simple linear regulator. It requires that a bipolar symmetrical power supply is connected to its power input. Then the output voltage is proportional to its control voltage input. The maximum voltage output is rated at +-10V. The maximum current is rated at 1A. No constant current source/sink capabilities were listed. Given that the power supply is not included, the price is (in my opinion) inexplicably high at 1880czk as to 12.11.2016 However, it's possible that massive discounts would apply if we attempted to buy large quantities.

- relay module

A module consisting of a single relay (RP210 12V/2P). Given the module is probably literally just this relay in a box (and a flyback diode, given the remark that doesn't switch on negative voltages) [31] and can be bought online for around 30czk [32], the module price of 1180czk seems bit unjustified.

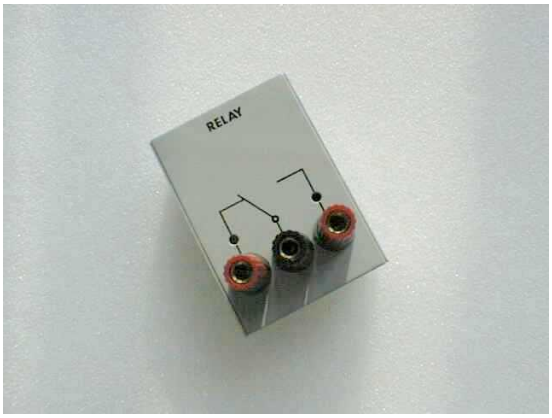


Image 4 - the ISES platform relay [31], [33]

- speaker

  A simple speaker output module - not much to be said about it. Includes volume control

Openness

ISES is an open platform, making it also compatible with similar products from Vernier, Pasco, etc. [28] Furthermore, the ever-more-popular Arduino devices can be used with this platform, which is perfect. They can be used as analog voltage sensors and the GPIO can be accessed; the required Arduino firmware comes in 2 versions - lite, which only allows one analog channel of the Arduino to be used and a full version, which is without restrictions.

Conclusion

Overall, the ISES platform is perfect for what it was created for - that is education. When purchased by schools, the higher price of some components is not really an issue, or may not really be the case due to large quantity orders. It is not intended for automation of any kind and certainly isn't a good purchase for an electronics hobbyist. Thus, while there are some particular applications where both ISES and my platform could be used as a solution, the target demographics and intended uses are vastly different.

Vernier

Introduction

Vernier is a measurement and analysis toolset, aimed at education, similar to the aforementioned ISES.

User interface

Overall, its user interfaces seem more professional than that of ISES; the PC GUI is funnily named "Logger" and comes in 2 versions - pro and lite. [34], [35] Lite version is free (!) and supports basic data collection and display. Logger pro then supports online data analysis and has a few additional feature, such as the ability to fit a function into measured data. The logged data can be stored.

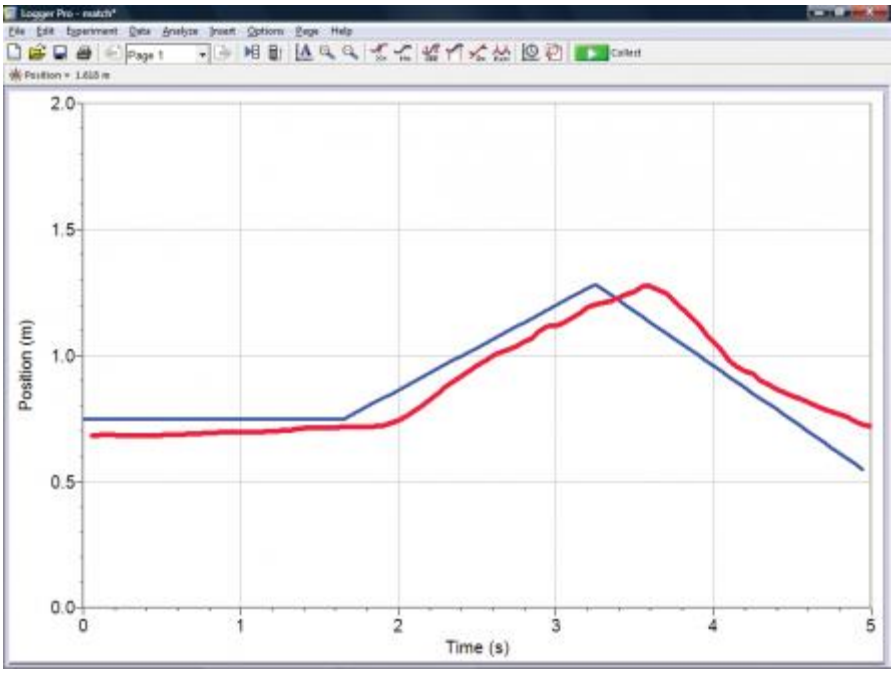


Image 5- Logger PRO user interface [34], [36]

The user interface seems to be modular to some degree, but overall, ISES wins in this aspect.

Automation

As for the automation and self-containment: specialized hardware dataloggers (so called LabQuests) exist that allow sensors to be plugged into them. Functions of such dataloggers seem to be rather advanced; however, their price is rather large (16970 Czk for the cheapest device with display). [37]

Image 6- LabQuest2 PDA [37], [38]

The PC-side GUI has a nice look and feel to it and also supports a certain degree of automation in the pro version. Simple conditions (such as value being within a certain range) can be checked and actions can be executed accordingly. Keyboard shortcuts exist (albeit ones quite counter-intuitive) to speed up the workflow, but carrying out repetitive tasks seems to be cumbersome, even based on the videos provided by the manufacturer itself.

Output and control

LEGO sensors can be connected to the Vernier platform over an adaptor. [39]

Regarding the output modules: there is a power amplifier module available. The properties are similar to the ISES booster module, at least on paper - the output voltage range spans from +10V to -10V, the maximum output current being 1A. Frequency wise, the maximum supported range is 10khz. The price is 12226czk as to 12.11.2016.

On the official Vernier product list, I have also found an Extech power supply, but no signs of compatibility with the rest of the system are hinted. [40] I presume it's a power supply that can be used to power some of the modules, but cannot be controlled digitally by the platform - this is in accordance with the Datasheet of the supply provided. [41] Some of the sensors produced by Vernier can be connected wirelessly to a tablet or a smartphone.

Openness

The openness of the Vernier platform is really good, as Vernier has support for the Arduino platform and allows the connection of various Vernier sensors directly to Arduino. [42]I haven't found any information on whether it's possible to do this vice versa, e.g. connect an Arduino to the Vernier dataloggers, for instance, and use the Arduino as a sensor.

Vernier sensors are also compatible with the ISES platform, mentioned above.

Conclusion

Overall, the Vernier toolset offers many devices and can be an interesting choice for a school. Some of the simpler solutions offered by Vernier are very inexpensive to the point

an electronics hobbyist could afford them. This being said, the product list consists mainly of sensors of non-electrical quantities, such as thermometers or PH-meters, and not so much on electronic lab equipment. In conclusion, we have found another great tool for education, however, for home use by an electronics hobbyist, the price is still prohibitive and the focus of the platform is just different, meaning not many modules such a hobbyist would utilize exist, apart from a very low range voltmeters with the highest range of +-30V, hall effect current sensor and a low range (600mA) ammeter. [40]

Pasco

Introduction

Pasco is another company, making product aimed primarily at the education. The sensors get connected to the main module and over it to a PC. All the versions of the main module, apart from the highest one, allow the connection of the maximum of 2 devices to them. The highest version of the module, named "PASCO 850 Universal interface", which resells at 53850Czk as to 13.11.2016 allows the connection of up to 4 devices. [43]

Image 7- Pasco 850 universal interface [43], [44]

It needs to be noted that it has some additional features apart from allowing the sensoric connection: it can also serve as a (very limited bandwidth of 500khz) oscilloscope, 100khz dual channel function generator (with a surprisingly high power output of 15W (!!) and max voltage/current capabilities (!!!)) and it also has 4 analog inputs, complete with a rugged input protection. [43]

Adaptors are available, that allow the connection of a single sensor to a pc and present no additional functionality. Previously an USB sensor adaptor existed; it has now been deprecated and replaced with a universal adaptor which also allows sensor to be connected to the PC via the USB interface, but additionally also allows a wireless connection over Bluetooth. This adaptor is called AirLink. [45] Instruction videos available on Pascos page

indicate that it is possible to connect multiple sensors to a tablet this way, utilizing multiple instances of the AirLink adaptor. [45] Airlink currently (13.11.2016) resells for 3200Czk. [45]

User interface

Two main pieces of software exist that support the data collection from Pasco sensors:

SPARKvue

This software works on many different platforms, ranging from Android over IOS all the way to Mac and Windows. [46]Linux support is not listed. Special dedicated datalogger tablets (SPARK Element) are sold by Pasco, that allow the connection of many of the sensors sold - those run Android 5.0 and have the SPARKvue software preinstalled. [47]

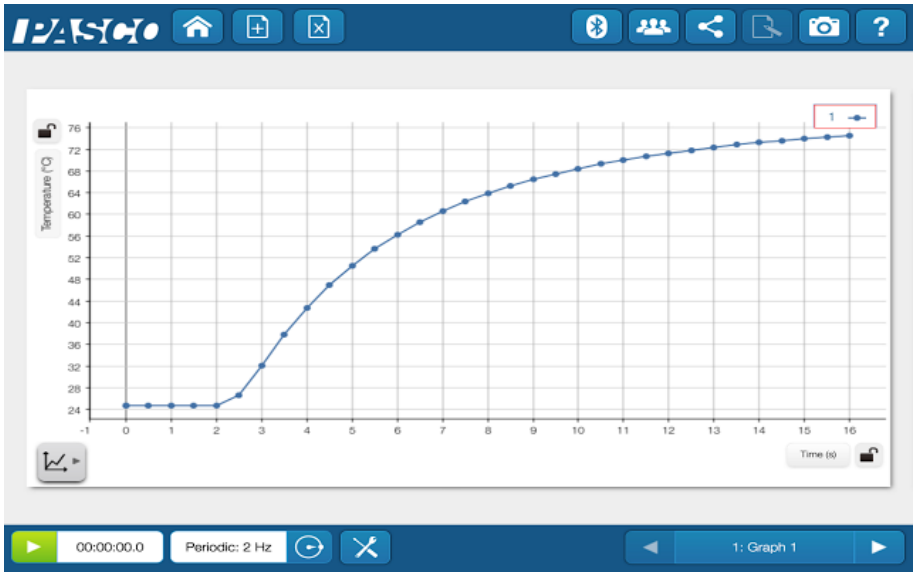


Image 8- SPARKvue user interface [46]

This software allows a modular GUI creation using charts, gauges, digital displays and more. It is actually available free (!) for mobile devices and has a 60 day trial for PCs. It provides basic tools for data analysis and curve fitting. It has no support for online data processing or automation of any sort. [46]

Pasco Capstone

Pasco Capstone [48] seems like a rather powerful tool; it is available on PCs and Mac computers. It supports online data processing to a rather large extent; it even supports user defined functions. It allows the creation of a rather modular GUI. Both the measured and calculated data can be plotted in a regular chart, displayed on a gauge meter or a digital meter and processed to show a histogram or a FFT display (!). It is also possible to take multiple measurements and place them in the same chart. No support for automation seems to be present.

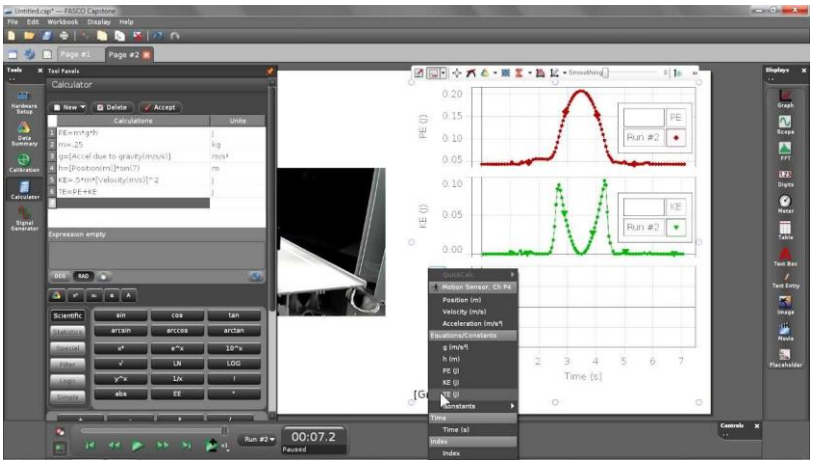


Image 9- Pasco Capstone user interface [48]

Automation

No support for automation seems to be implemented in any of the 2 pieces of software listed. This being said, it seems possible to use Pasco instrumentation together with the Labview software... [49]

Output and control

Some of the interfaces the "Introduction" section above mentions are capable of providing a constant current and constant voltage output, as well as generate an arbitrary waveform. Those can be controlled by the Capstone and presumably also the SPARKvue software.

Openness

Based on the papers available, the platform seems to be closed, although it doesn't explicitly state it anywhere.

Conclusion

The closeness and the high price of the hardware in question are the only (yet critical) flaws of this platform, that would otherwise be usable to an electronics hobbyist - options to output constant voltage and current (even as a waveform) exist, it is possible to take rather fast measurements of all sorts of quantities and overall I really wanted to like it. My other complaint would be the lack of support for scriptability and automation. Lack of a dedicated hardware base user interface (a panel with knobs) wouldn't be that much of an issue, thanks to the support of touch screen devices. Pasco products are something I'd buy myself for Christmas, did I have unlimited resources and income. They seem like a terrific learning and experimentation tool, that could even be utilized in research. Granted-I'd personally do some things differently, but all in all, those could be tolerated - and if the need to measure voltages over 30 volts and high speed signals doesn't emerge -the closeness of the platform and the price of its components remain the only -yet unovercomable- obstacles between it and an average electronics hobbyist.

Papouch

Introduction

Papouch is a company behind many interesting sensory modules, aimed mainly at industrial electronics monitoring and measuring technology. Their main area of expertise and focus seems to consist of thermometers and various humidity sensors [50] [51], but their offer includes relay switches [52], wind speed sensors and more. Some of their sensors can be connected over USB; those need no additional hardware. Additionally, interfaces exist that allow the connection of various sensors to them; the data measured can then be accessed over the Ethernet. The number of sensors that can be connected to such an interface obviously varies with the specific interface in question; the "TME multi" for instance allows taking measurement from up to 32 humidity and temperature sensors simultaneously. [50] Papouch products are mostly intended for remote monitoring of facilities and production runs.

User interface

The data collected by the sensors can be displayed using a simple web browser, which connects directly to the webpage, hosted by the interface. Alternatively, a piece of software called "Wix" can be used to display the values. [53] Some additional UI options exist, such as a Windows desktop widget, which shows the measured values on the desktop.

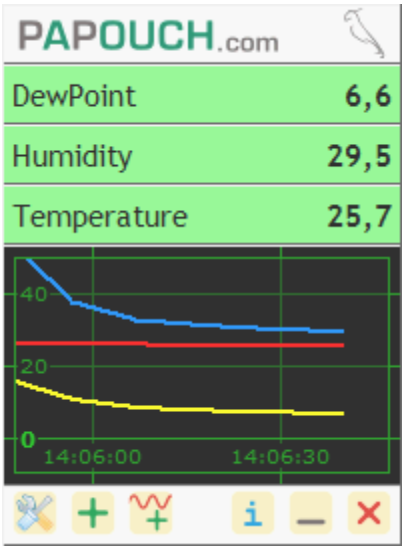


Image 10- Papouch widget [53], [54]

Remark: When searching web for the screenshots to put into this document, I have noticed that Papouch now offers special Ethernet-enabled sensors, that can reply to POST or GET html requests and the value read by them can thus be integrated into any webpage. [51] Data processing could then be written in languages like PHP or Javascript by the user. [51]

Automation

Some degree of automation is possible with the Wix software, which is available free for smaller numbers of measurements. Conditional clauses can be written such as "if temperature goes above 40°C, turn on the fan". This automation is limited, although probably fully adequate for most intended usage cases. The Wix software included doesn't come with any support for an online data processing (then again, I can't think of any usage case Papouch products were intended for, that would benefit from this).

Remark: allegedly, this is now possible as a part of a paid service from Papouch.

Output and control

Modules exist that allow a relay to be switched remotely. That's about the extent of output for Papouch devices.

Openness

Papouch products are not open by design, but given how simple some of the sensors are, they would be very easy to hack.

Conclusion

Papouch products are a great inexpensive choice for measuring temperature and humidity; they aren't very well suited for measuring voltage, current or related quantities. The software provided is rather basic and doesn't provide any online analysis toolset.

myDevices Cayenne

Introduction

MyDevices Cayenne is an interactive IOT platform builder project. [55] It allows users to visualise and collect data measured by Arduinos, Raspberry PIes and other devices, connected to the Internet, with a special software running on them. [56]

User Interface

2 GUI programs are a part of this platform: one that runs in a web browser and one for mobile devices. Those programs allow the creation of a GUI, mostly by selecting which of the values measured do we want to display and how (chart/gauge/digital display). Values can also be set using this GUI. [55]

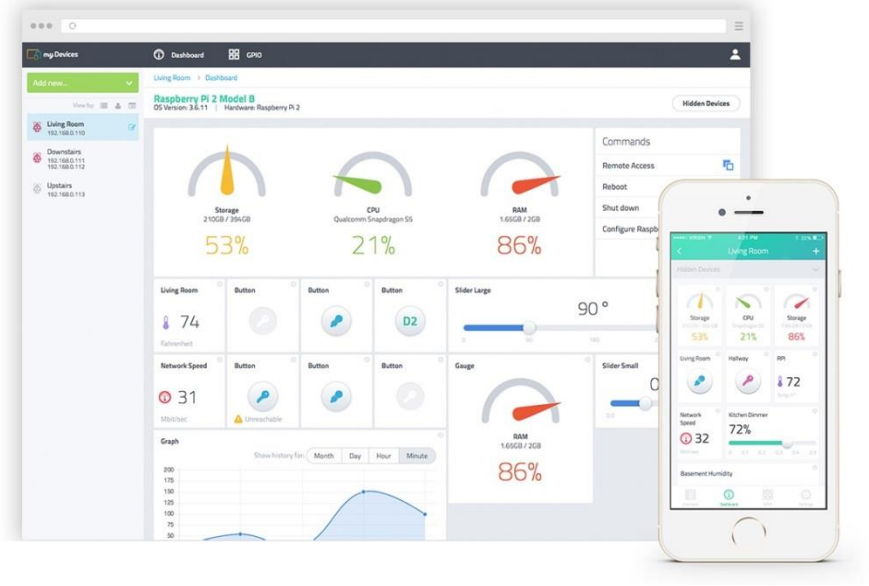


Image 11- myDevices Cayenne user interface [57], [58]

Automation

So called "Triggers" can be setup to decide what should happen if a condition is met. [59]The conditions are very simple, such as "value is over a certain threshold". The actions can either have the form of a notification or some action of a certain device connected. The triggers can span over multiple devices. For instance, it is possible to setup the platform in such a way, that when Raspberry PI senses the temperature is above a certain threshold, Arduino connected to the Internet in another city blinks an LED.

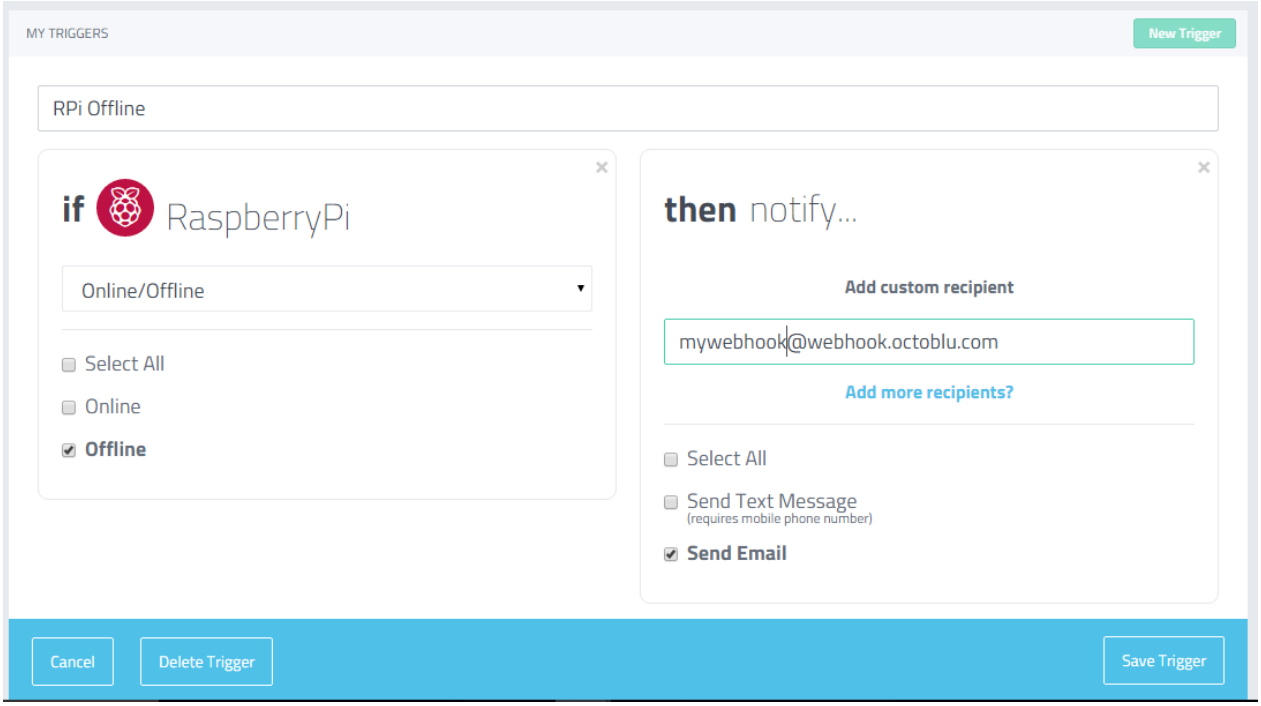


Image 12- myDevices Cayenne triggers [60], [61]

Output and control

It is possible to send analog values to the connected devices over the GUIs and handle the behavior of those devices in their code accordingly.

Openness

MyDevices Cayenne is a truly open platform. It supports many open source and partially open source hardware pieces, including Arduino or Raspberry Pie and can control those remotely as well as connect to them. [56]

Conclusion

Currently (13.11.2016), myDevices Cayenne is still in beta. It is free, but supposedly will not stay that way forever. Payment models have not yet been decided according to the helpdesk forums, but I suppose a monthly payment subscription models will be adopted.

Since all the data processing is handled on the server side, there is a considerable lag between a measurement being taken and it reflecting on the screen. Since the data is collected from easily programmable and beginner friendly devices, such as Arduino, the code uploaded to such a device can be altered to handle data processing on the device (Arduino) side. However, it goes without saying that this is not an interactive process.

It allows some minimal degree of automation and data processing. Its main strength is the access from anywhere and also the fact multiple target devices can be accessed at the same

time. The main downside I see is the slowness of response and the dependence on the main Cayenne servers and Internet connection. This product, while indeed a very good and useful idea and a nice toy, is unsuitable for say, quick subsequent readings of a voltage, due to the ping sometimes being around 0.5s, even on their own videos. [62]

Graphical User Interface documentation

Introduction

I have designed the Graphical User Interface (GUI) in Java. The main design goal was for it to be very flexible, scriptable and mostly controllable from a keyboard.

The requirement to be able to control most of the functionality from the keyboard was introduced for 2 reasons: First, I have found myself to be much more proficient when using keyboard-oriented user interfaces than when using mouse-oriented ones. Additionally, at some point in future, I would like to port the controlling application, which now runs in PC, to some embedded platform, such as Orange Pi. Then the main module would be formed by a dedicated hardware, as opposed to a PC and the whole system would be standalone. This system then would be controlled using a set of physical user interface (UI) elements, such as switches, buttons and knobs, located on the panel of the main module. I figured that focusing on keyboard-oriented control from the very start will eventually make migration to this new UI paradigm easier.

The CLUC language compiler/interpreter, designed and written by me in Java, is closely intertwined with this GUI application. Different properties of the GUI can be edited by calling CLUC functions, and editing them “manually” can optionally fire the execution of CLUC functions, too.

The GUI of my instrumentation platform project was designed to resemble the user interface of the Vi and Vim text editors to the largest possible extent practical. in some cases, it was not practical or possible to handle things in the same way as Vim does. In some cases, functionality that simply isn't present in Vim needed to be implemented. In this document, I'd like to start with a brief overview of how Vim behaves and then I'd like to cover some of the aspects of my GUI, making remarks towards how well the particular feature adheres to the Vim paradigm.

What is Vim?

Vim and its predecessor Vi are very popular modal text editors. Modal simply means that they have various different modes that the user can switch between. The mode in which

Vim behaves like a standard text editor is called the "Insert mode". In this mode, the letters pressed get typed on the screen.

There are several other modes, but for the sake of simplicity, without resigning on completeness too much, we will just list one more: the so called "Normal mode". This is the default mode - all modes can be exited into Normal mode by hitting "escape," albeit sometimes more than once. In this normal mode, the keys pressed have special actions assigned to them. For instance, typing "dd" deletes the current line; typing "gUe" "Go Uppercase till End" capitalizes the word following after the current cursor position, etc. Pressing "a", "i" and some other keys enters the insert mode, while placing the cursor on a certain location, based on which key was actually pressed.

By design, Vim is intended to be used in the insert mode as sparely as possible. The main philosophy behind Vim is "you spend more time editing text than actually writing it".

This approach has several benefits. One of them is that the keyboard "shortcuts" tend to mostly have the form of several keys typed one after another, instead of a wild combination of modifier keys and letters. This often makes the shortcuts easier to memorize. For instance, the command to delete text from the current cursor position to the end of word would be "de" - "Delete till End". In the (also popular) text editor Emacs, one would need to hit "alt-d". Additionally, the users hands need to move from the home row in order to hit the modifier keys and hitting the correct combination is therefore slower and less comfortable than typing stuff in.

The modality is just one great aspect of the Vim UI paradigm. There are many more - but since the aim of this document isn't to cover the Vim user interface, I will just mention those briefly where relevant, that is, when describing features of my GUI that are directly or indirectly based on the properties of the Vim text editor.

GUI layout

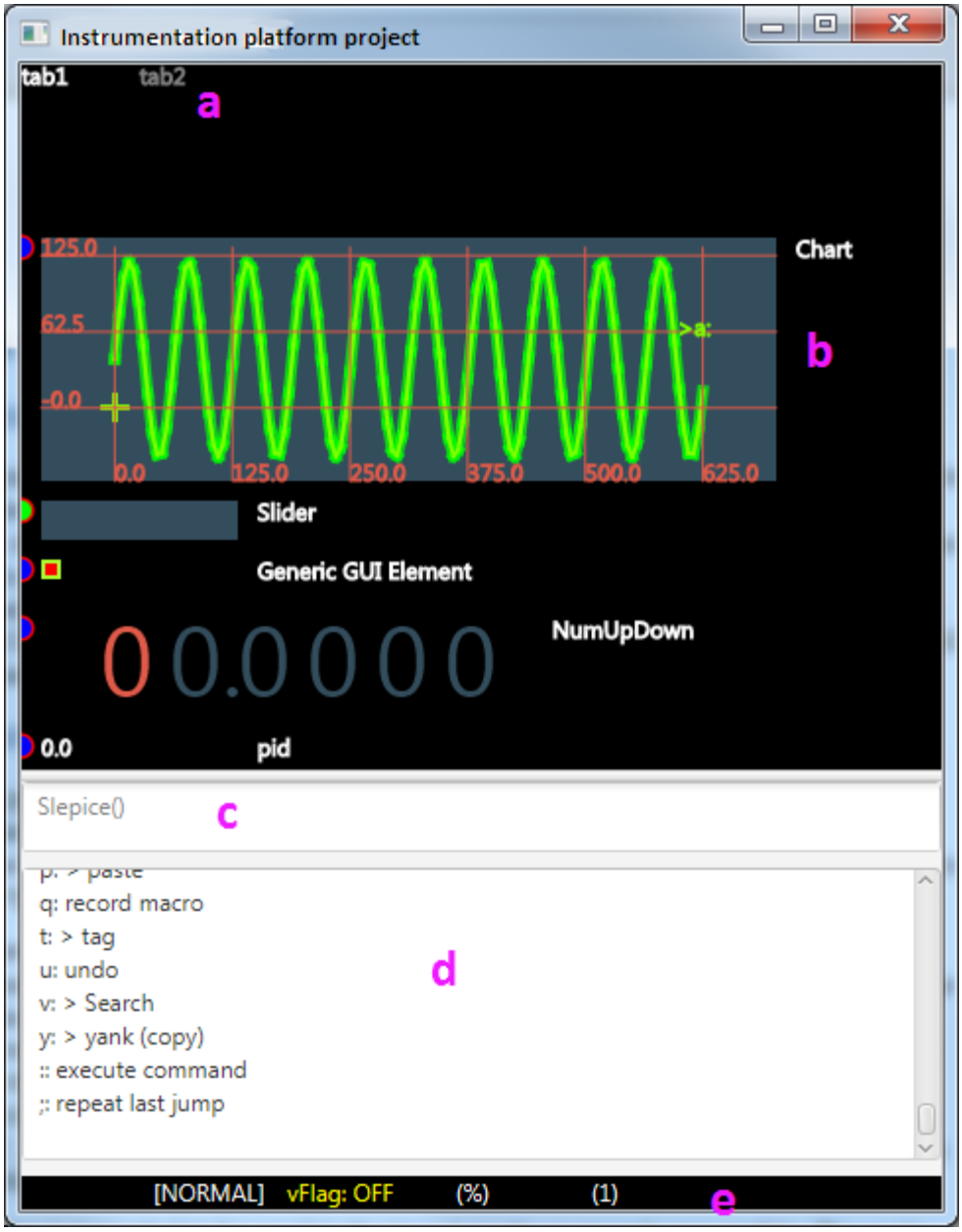


Image 13- GUI overview

The main GUI window can be divided in 5 main parts:

a) Tab header. The individual GUI elements, such as a chart, slider or a display, are placed on different tabs. These tabs are similar to the tabs in Vim or in any modern Internet browser. Only the elements placed on the currently selected tab are displayed. The tab header section gives the user an overview of the tabs present and also highlights the currently active tab.

b) Tab content (panel). This is where the content of the current tab is displayed. Different user elements can be accessed from here.

c) Command line. This line can be used to immediately and interactively execute code snippets, written in the CLUC language. Such code snippet usually only spans a single line, but multi-line code is allowed. This line can be used to change the properties of individual GUI elements or to call user functions. It can also be used for non-code user input, such as entering strings or numbers when prompted by the GUI.

d) Status window. By default, this window displays a list of different commands available, complete with the key presses needed to run this command, lowering the cognitive burden for the user. This window can also display error messages, autocompletion suggestions for the command line, or results of calculations.

e) Status line. This line shows useful information to the user, such as alerting them that an error occurred.

There is also a secondary window, which serves as a simple User code editor. The user code is a list of CLUC functions, which the user can provide, that can then be called from one another and also from the command line. Naming the user functions according to a specific convention, which will be covered later, allows those functions to be called automatically by the GUI, for instance, whenever some property of a certain GUI element changes.

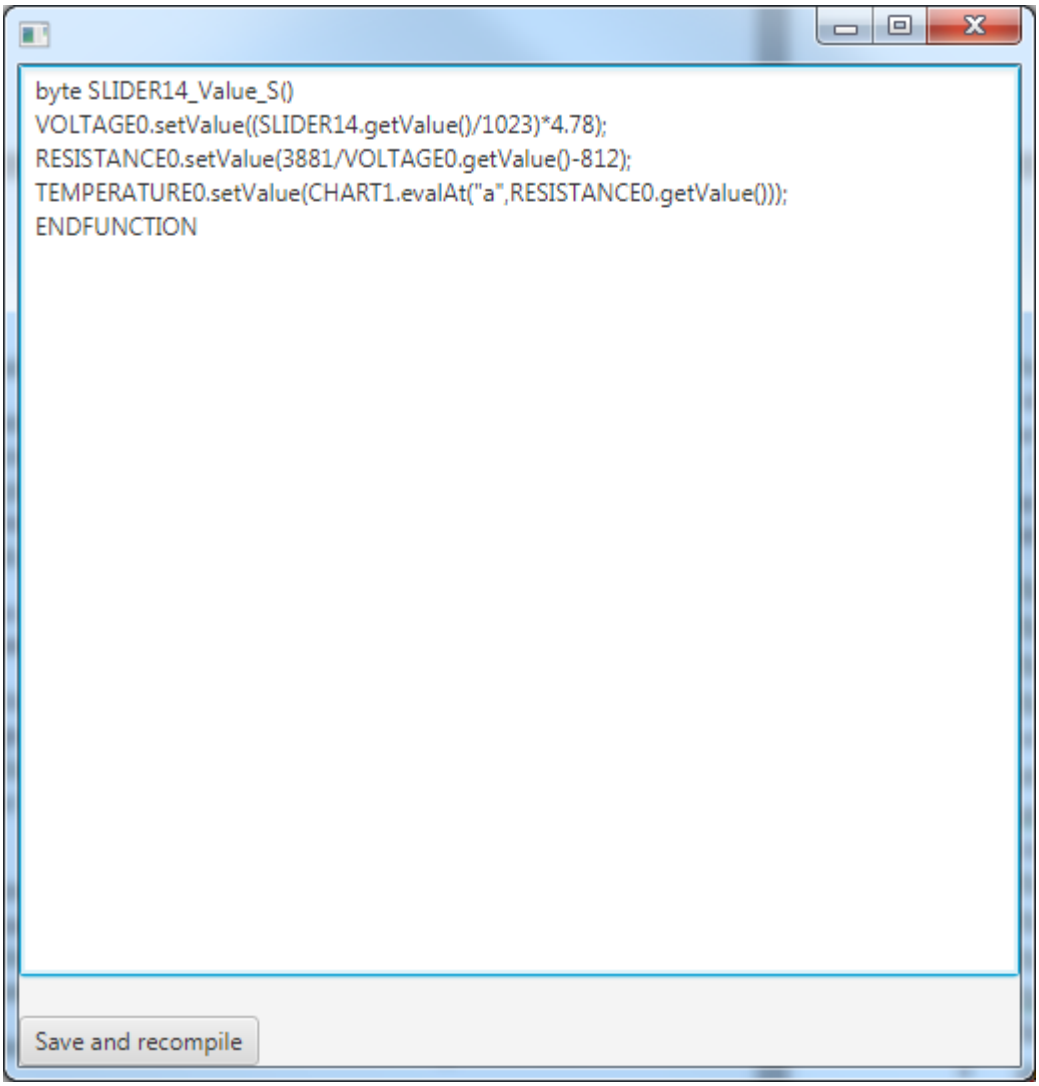


Image 14- Built-in user code editor

GUI elements

Element focus

Since there is an advanced selection mechanism implemented in the GUI, allowing multiple elements to be selected at once, we need to pick a different word to denote that a certain element is picked from the list, ready to be edited. Let's refer to this as the element being focused. Only one element can be focused at the time, so focusing an element implicitly means defocusing all other ones.

To summarize it: multiple elements can be SELECTED at once and then some action can be executed, which can affect all of them. However, just one element can be focused.

Properties

The building blocks of the GUI, that is various displays, charts, sliders, etc. are referred to as "GUI elements" or “GUI components”. All those elements have certain properties. Those properties can be accessed from certain commands and user code, which will be covered later. There are some common properties that each type of GUI element has; some properties are specific for a given element. Here is the list of properties common for all GUI elements:

- float Value the current value set
- float Max the maximum value that can be set
- float Min the minimum value that can be set
- float Step the minimum step with which the value can increment; if we have a display with 2 digits after the floating point, this value would be 0.01
- string Name the human readable name of the element
- string UniqueName the (not so readable) unique name of the component. Must not contain spaces and special characters. Is usually automatically generated.
- float Color1
- float Color2
- float Color3
- float Color4 Various colors for the element. 4 bytes stored in the RGBA format, 0-255 for each channel.
- int Focused whether or not is the element currently focused
- int Selected whether or not is the element currently selected
- int Visible whether or not is the element currently visible
- int Width the preferred width of the component (in pixels)
- int Height the preferred width of the component (in pixels)

All of the properties listed above can be adjusted by the module in real time. Any module can also make a request to be informed, whenever any of the properties change (and get the new value sent to it).

They can also be accessed by the user from the user code, or from commands ran from the command line.

Syntax to achieve this is following:

GUI_ELEMENT.setProperty(value);

and

GUI_ELEMENT.getProperty()

Where GUI_ELEMENT is the UniqueName of the element, property of which the user wishes to read or modify and Property is the name of the property, starting with a capital letter.

Additionally, the “CGE” string can be used instead of the UniqueName. CGE stands for “Current GUI element” and when used, the meaning is simply “the currently focused element.”

Example:

```
SLIDER5.setValue(200);
```

to set the Value property of SLIDER5

or

```
DISPLAY0.getHeight()
```

to get the Height of DISPLAY0.

Additionally, there are functions that can be called on the GUI elements. Function calls have similar syntax; for instance:

```
GUI_ELEMENT.sendKeys("abcde");
```

Some GUI elements, such as a Chart, have additional functions that can be called on them; however, here is a list of functions common for all elements:

- int sendKeys(string keys)

Has the same effect as focusing the given component, editing it and typing the keys in the string given one by one, before hitting (esc) and focusing the previously focused component.

- int hasTag(string tag)

Whether or not is the given component tagged with the specific tag. Returns 1 if it is, 0 if not.

Callbacks and user code

Each element also can call specific user functions (functions written in my language by the end user) whenever something happens to it. Those functions are known as "User interface callbacks". The list of such functions, that all the components have is below:

[UniqueName]_[PropertyName]_S()

Setter callbacks.

Group of user functions, which get called automatically, whenever a property is changed.

One such example would be SLIDER1_Value_S() which would automatically get called once the "Value" property of the element SLIDER1 changes.

The “S” in name simply stands for “set”.

[UniqueName]_[PropertyName]_G()

Getter callbacks.

Group of user functions, that get called automatically, when a property is read.

One such example would be DISPLAY1_Value_G() which would automatically get called once the "Value" property of the element DISPLAY1 is read from.

The “G” in name simply stands for “get”.

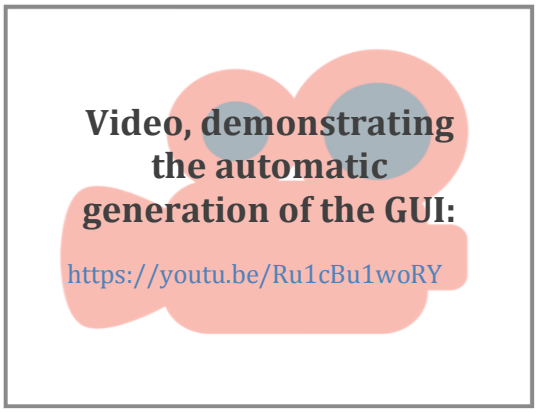
Getter callbacks are not intended to be used very often; however, omitting them would introduce unnecessary imbalance.

Some internal processes of the GUI, such as displaying a value of a display on screen, may also read a property, without firing this callback.

Additionally, some elements have their own callback functions. Those will be listed in the subsections of the [GUI elements] section, dedicated to the respective elements.

Default GUI

Once the GUI program is run and child modules are plugged to the PC, the user can give a command, which will establish a connection to a child module and then ask this module to provide a description of a default user interface, usable to collect data from this module, and to control the module outputs, are there any.



This description of a user interface includes the type of the individual user interface elements, their names, whether they can be

written to, read from, the minimum value, maximum value, colors and more. These properties can be set by the module during the creation of the user interface, or at any other point in time later. Any child module can update all the properties of its GUI elements. This means, for instance, that a power supply module might restrict the maximum value of a slider that sets the voltage, once the current drawn is too high, or that a temperature sensor can automatically change the color of the temperature display to red whenever critical.

The GUI is divided into tabs. Each module can also request the creation of a new tab and then place its GUI elements on it.

Although it's not a requirement, it usually makes sense for each module to create its own tab and then place its GUI elements on it.

This resulting GUI is fully usable, although not necessarily the most convenient.

For this reason, it can be edited by the user by hiding some GUI elements, moving them around (even between tabs) and adding new ones.

Keyboard shortcuts concepts

Just like in Vim, the keyboard shortcuts have the form of several keys pressed subsequently (typed). Thus, when a shortcut "gg" is mentioned in this document, it simply means that "gg" has to be typed to execute the corresponding action (which happens to be focusing the top of the GUI element list in this case). Apart from that, there are 3 ways to pass arguments to the actions executed:

- typing a number before executing the command this results in this number being passed to the command in question. Some commands ignore it, but most take an (optional) numeric argument. The typical meaning of the numeric argument is "repeat the action N times". Thus, typing "j" focuses the next element (after the currently focused) and typing 2j focuses the 2nd next element. Once the action executes, the number is reset. This is consistent with Vim.
- typing the >"< (single quotation mark) key, followed by a letter sets this letter to be used with the command to follow. In vim, typing >"< followed by a letter denotes that the next delete, copy or paste action should be associated with a register of a given letter. For instance, typing "adaw in Vim deletes a word (daw) into register a ("a). To paste from this register, the user can type "ap (from register "a" paste) This way it's possible to access numerous registers, which serve as a sort of clipboard, a place where pieces of text can be stored.

This principle is built upon in my GUI and it is extended. The support for registers is present and the way to access them is similar; however, single letter arguments can be passed even to commands that won't use it to access a register. One such example would be the mark command, where hitting "am assigns the currently focused GUI element the mark "a"; this mark can then later be used to focus this particular element once it loses focus.

- pressing "V" (capital v) toggles the so-called vFlag between on and off. The current state of vFlag is reflected in whether the next command executed should apply to the whole selection (vFlag on), or to the currently focused element only (vFlag off). Attempt to run a command with the vFlag on and no components selected is reported to the user as an error.

The bottom part of the GUI contains a multi-usage status window. One of the usages is listing all the commands currently available. All keyboard shortcuts are treated as a menu and hitting keys traverses it. Thus it is not necessary to remember all the shortcuts. In case of doubt, all it takes is to take a look at status window. For instance, by default, the text box lists that hitting "g" will enter a "go to" submenu. Once we press "g", the textbox reflects it by informing us that subsequent press of "g" will let us go to the beginning (focusing the first element in the tab), press of "t" will go to next tab (switching to it), etc.

This is a concept not present in the Vim text editor, but when I invented it, I found it useful, thus I implemented it.

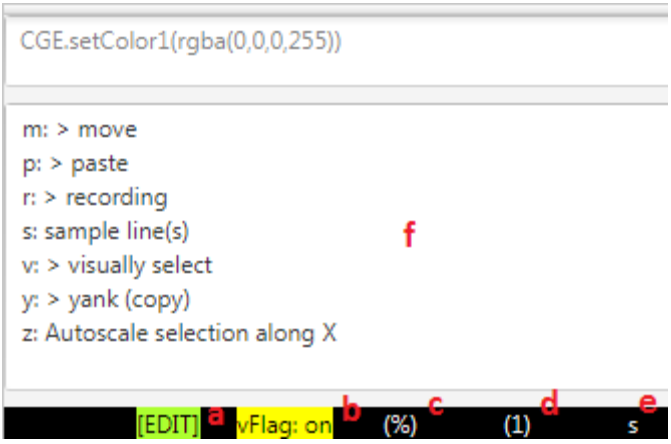


Image 15 - status line and status window

This picture shows the status line and the Status window

a) current mode (is the user editing a component?)

b) vFlag status

c) currently selected register

d) number, which should be passed to the next command executed

e) last pressed key sequence

f) status window

In the text to follow, the "N" before the motion suggests the command accepts a numeric parameter. the "L" suggests it accepts a letter parameter. [V] before the command then suggests in can also be applied to a whole selection when the vFlag is on.

Action concepts

There are 2 types of user actions; one which modifies something and one that just changes which element is focused. The first one can be referred to as edit; the second can be referred to as jump. The list of elements that had focus recently is being kept and is known as a jumplist. Additionally, the specific jump that was executed last is also being tracked. The list of all recently edited elements is kept as well; this is known as changelist. This is similar to the jumplist and changelist concepts in Vim.

Basic motions

Here is the list of basic motions and motion related commands that the GUI supports. By motion we simply mean an action, main purpose of which is focusing a different GUI element. Basic motions are all jump actions.

- N j focus Nth next element after the currently focused (go down) . Jumps focus to the next element by default.
- N k focus Nth previous element after the currently focused (go up) . Jumps focus to the previous element by default.
- N gg focus Nth element from the top, counting from 1. 1 by default.
- N G focus Nth element from the top, counting from 1. The last element in the list by default.
- L m mark the currently focused GUI element with the letter L. Remove the mark from the element which had it, if there was any. If L is a capital letter, this mark is global. If not, it is local. See the next line for more info.
- L ' jump to the GUI element (focus it) marked with the letter L, if it is on the same tab as the current one, no matter what and focus it, changing the tab, if its mark was global (that is, it is a capital letter).

Tab motions

Tab motions are also jumps.

- N gt go to next tab (right from the current one) N times
- N gT go to previous tab (left from the current one) N times

Jumplist

- N o jump to previous (older) location N times, focusing the previously focused element, switching the tab if needed.
- N i jump to the next location N times, opposite of above
- N ; (semicolon) repeat the last jump action N times

- N , (comma) repeat the inverse of the last jump action; for instance, if the last action was "2j", the inverse would be equivalent to typing "2k"

This is, more or less, consistent with Vim. "o" and "i" were used instead of Vims ctrl+o and ctrl+i, as the keys o and i aren't reserved here.

The scope of ";" and "," is a single line in Vim, but that isn't applicable here.

Change list

Traversing the change list means focusing the GUI elements in the order they were edited.

- N gi focus the element that was edited the last and start editing it non-interactively
- N gI focus the element that was edited the last and start editing it interactively
- N g; traverse the changelist backwards, first focusing the element that was edited the last; then focusing the one which was edited before, etc.
- N g, traverse the changelist forwards

These actions are also jumps.

Undo and Redo

(The undo system has been implemented, however, further changes rendered it temporarily unreliable and it has therefore been disabled. It needs to be fixed to also support undo trees, selective undo and persistent undo.)

It is possible to undo the edit actions and then redo them.

- N u undo last N edit actions
- N r redo last N edit actions
- N g- roll the state of GUI elements back historically N times
- N g+ opposite of g-

Additionally, it's also possible to repeat the last action

N . repeat the last action N times

Tags

Each user element can have a set of tags assigned to it. One element can have multiple tags assigned. Those tags have the form of different letters. Tags can then later be used to limit an action, which would affect multiple elements, only on the elements with the given tag.

Since it is possible to select all components with a given tag and also to tag all selected components with a letter, tags can also be used to “store selections.”

As far as I know, no similar functionality is present in Vim, however, a distantly similar functionality was implemented in Ranger, the Vim-inspired file manager.

- [V] L tt tag the currently focused GUI element ([V]=all selected)with the letter L. When the element already is tagged with this letter, remove this tag. When no L is provided, remove all tags.
- L tv visually select all GUI elements containing the tag L.

Names

As mentioned above, the GUI elements have 2 names: the "human readable" one and an Unique one. No 2 GUI components can have the same Unique name (thus it's called unique). The unique name cannot contain characters different from underscore, uppercase letters and numbers and cannot start with a number. The Unique name contains a legal form of the human readable name (converted to uppercase and stripped off illegal characters), the Generic name of the GUI element ("SLIDER", "DISPLAY"...) and sometimes a number, in case the previous wasn't enough for the name to be unique.

Whenever we will be referring to the Name in the text to come, we could be refering to both the Unique and the human readable names; which one of those it is depends on the current settings.

- gU switches the current context to unique names
- gu switches the current context to human readable names

This changes which one of the names gets displayed to the right side of each GUI element, but also affects all searches (as covered in the next section) and more.

Filtering

It is possible to temporarily hide GUI elements, making them invisible, in order to make the current user interface less cluttered. Hiding an element doesn't prevent it from being updated or its changes to propagate to the module.

Some of the commands listed below require a string. Once the command is executed, the commandLine on the bottom of the GUI activates, prompting the user to enter the given String. There are 3 types of strings that can be used: a literal string, a regex (regular expression) and a fuzzy String. As the string is being typed by the user, the elements that will be affected once "enter" is pressed are getting highlighted in real time to preview the action that will take place.

Image 16- filtering preview

The search will take place between the unique names or human readable names of the GUI elements in question, depending on the current settings. This makes it possible to perform filtering based on the module connected to the GUI element, the type of the GUI element and more. (See the above section "Names" for more info.)

- [V] fc hide the currently focused GUI element or all selected
- fr filter using regex (only show the ones containing the regex, hide the ones that don't)
- fR filter using regex (only show the ones NOT containing the regex, hide the ones that do)
- ff filter using fuzzy search (show the ones containing the fuzzy expression, hide the ones that don't)
- fF filter using fuzzy search (only show the ones NOT containing the fuzzy expression, hide the ones that do)
- fl filter using literal string (show the ones containing the literal string, hide the ones that don't)
- fL filter using literal string (only show the ones NOT containing the literal string, hide the ones that do)

The filters can also be additive:

- f+r filter using regex (show the ones containing the regex, DON'T hide the ones that don't)
- f+R filter using regex (show the ones NOT containing the regex, DON'T hide the ones that do)
- f+f filter using fuzzy search (show the ones containing the fuzzy expression, DON'T hide the ones that don't)
- f+F filter using fuzzy search (show the ones NOT containing the fuzzy expression, DON'T hide the ones that do)
- f+l filter using literal string (show the ones containing the literal string, DON'T hide the ones that don't)

- f+L filter using literal string (show the ones NOT containing the literal string, DON'T hide the ones that do)

Or they can be subtractive:

- f-r filter using regex (hide the ones containing the regex, DON'T show the ones that don't)
- f-R filter using regex (hide the ones NOT containing the regex, DON'T show the ones that do)
- f-f filter using fuzzy search (hide the ones containing the fuzzy expression, DON'T show the ones that don't)
- f-F filter using fuzzy search (hide the ones NOT containing the fuzzy expression, DON'T show the ones that do)
- f-l filter using literal string (hide the ones containing the literal string, DON'T show the ones that don't)
- f-L filter using literal string (hide the ones NOT containing the literal string, DON'T show the ones that do)

Please note that it isn't possible to extract all specific subsets of the list of GUI elements on the current tab using the commands listed above. Namely it is not possible to automatically show or hide elements that contain BOTH regex A AND regex B. This doesn't mean such selection would be impossible; it is, indeed, possible, but the way to do that will be covered in another section ([Executing code snippets]), as the matter is a tad bit more complex.

Selection

All GUI elements can be selected; some commands can then be limited to only operated on the currently selected elements.

- vv toggles the selection of the currently focused GUI element
- va selects/unselects all on the current tab

For other commands that can be used with selections, refer to the "Filtering" section above. All commands used for filtering also have their version for selection. The first letter of such command isn't "f" but "v". The rest remains unaltered.

Registers

Registers in Vim are places, where text can be stored. Each is assigned on character, usually a letter. The user can copy text into those and then paste it back from them. They can be thought of as named clipboards. This GUI has support for registers. This usually has the form of pressing "(letter) (That is, a quote sign, followed by a letter) and the action related to that register, such as

copy or paste. This is described in different sections. It is possible to manually fill a certain register:

- L rf fill a register with string, provided by the user.
- L rc [L2] copy the content of L into L2.

There are 26 general purpose registers available, one for each lower case letter.

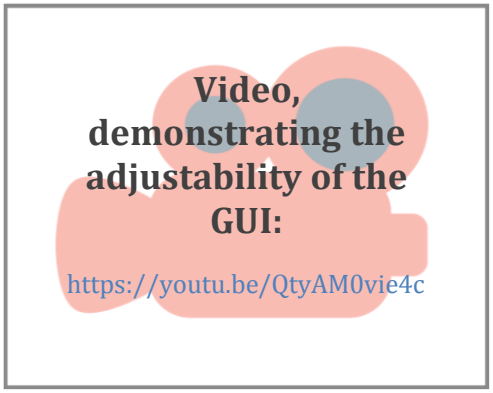
Whenever a change operation to the register is performed (one that changes its content), the new text is either placed there instead of the original one, or it is appended on a new line. Typing "(lowercase letter) before calling the commands overwrites the old content and "(UPPERCASE LETTER) appends to it. This is consistent with Vim. Apart from those 26 general purpose registers, some more are present

- % unnamed register; writing into it will store the text in the system clipboard
- + like above, but writing will APPEND the text to the text in system clipboard
- _ blackhole; writing to it will do nothing; reading from it will return an empty string. Since the registers() command can be used to display the content of all registers, being able to get rid of some text without the side effect of storing it into a register is handy, as it makes the output of this command less cluttered.
- [0-9] When the content of unnamed register is overwritten, the old content gets stored to register 0, content of which shifts to register 1, etc. Content of register 9 gets lost. These registers can be used to retrieve the old content of the unnamed register, when it was overwritten.

Moving the elements

The real power comes with the adjustability of the user interface. It is possible to move the individual GUI elements on the list, as well as between tabs.

- [V] L de deletes the currently focused GUI element from the list (or [V] deletes all selected), storing its uniqueName into the register L. Stores the name into the system clipboard if L wasn't specified. The GUI element object still exists, but isn't accessed and changes to its attributes don't get sent to the modules, until it is placed on one of the tabs.
- [V] L ye store the uniqueName of the currently selected GUI element into the register L. Stores the name into the system clipboard if L wasn't specified.



- N L pc makes N (default one) copies of the element, unique name of which matches the content of register L (or the system clipboard, if L is omitted). The new element has a different unique name. All of its other properties get disconnected from the corresponding module, meaning the module can no longer affect them. They remain in the state the original object was in when the copying action happened.
- N L pl (put link) places the element, unique name of which matches the content of register L (or the system clipboard, if L is omitted) below the currently focused element in the list N times (default one). No copy is created. Programatically, this doesn't create any other objects. Altering the properties of one such object will alter the properties of all links created. This tool should mostly be used whenever the need arises to have the same element on multiple tabs.
- N L Pc same as N L pc, but pastes ABOVE the currently focused GUI element
- N L Pl same as N L pl, but pastes ABOVE the currently focused GUI element

It should be explicitly noted that the information regarding which element was copied and which should be pasted is transferred using plain text. This technically allows for reorganizing the components outside the GUI, for instance using common text editors.

Accessing multiple values and other operations on multiple elements

It is also possible to copy the current value of the focused component or of the selected ones.

- [V] L yv copy the values of the focused component (or all selected components) into the register L, or system clipboard, if no L is specified, one on each line, in the order specified by the order of the given components in the list.
- [V] L dv before setting them to zero, place the values of the focused component (or all selected components) into the register L, or system clipboard, if no L is specified, one on each line, in the order specified by the order of the given components in the list.
- [V] L pv paste the values to the focused element from the register L, or system clipboard, if no L is specified. If [V] is on, paste the content of the first line into the first (topmost) element, content of the 2nd line into the 2nd element, and so on. User is informed about an error and the operation is aborted, when the number of lines doesn't match the number of selected components.

Accessing the element

Each of the elements can be accessed. When we are accessing a certain component, the actions that will be taken once we press buttons are different. This is similar to the modality of Vim. For instance, if hitting 50% previously focused the GUI element in the middle of the GUI list on the current tab, once a certain element is being accessed, it might, for instance, set its value to 50% of the maximum.

A focused GUI element can be accessed by hitting a.

- a - interactive edit; start editing the currently focused component; write each change to it (and the underlying module) as it is being made.

Hitting "a" would jump into edit mode in Vim.

Each element has a different set of shortcuts; however, those few are common for most of them; please note that those shortcuts only apply when a GUI element is being edited.

- c open prompt textbox, into which the user can type an expression. Once (enter) is pressed, the result of this expression will be assigned to the value of the GUI element in question.
- L yy copy (yank) the current value of the element into the register L. If no "L" was provided, copy into the unnamed register. Copying into the unnamed register has the side effect of storing the value into the system clipboard. If L was uppercase, append the copied value to the given register (named the lowercase version of L). If it was lowercase, overwrite the register.
- L dd delete the current value of the element, setting it to zero, storing it into the register L. If no "L" was provided, copy into the unnamed register. Copying into the unnamed register has the side effect of storing the value into the system clipboard. If L was uppercase, append the copied value to the given register (named the lowercase version of L). If it was lowercase, overwrite the register.
- L p paste the value stored inside the register L, setting it as the current value. if no "L" was provided, read the content of system clipboard and use that instead. If no valid value is in the register, ignore.

Additionally, each type of GUI element has its own keyboard shortcuts.

Remapping

It is possible to change the key mappings, that is, change the keys needed to do a certain action. This has the form of mapping one sequence of keys A to a sequence B. When sequence A is typed, the GUI "thinks" sequence B was. There are 3 types of mappings present.

Global mappings

Mappings, that are active no matter the current context

GUI element instance mappings

Mappings that are only active one a specific GUI element is being edited.

GUI element type mappings

Mappings active if and only if a specific type of GUI element (such as Slider) is being edited.

For more information on how to set up global mappings, see section "Global functions".

Macros

It is possible to record macros and then run them. Macro is stored inside a text register. When macro is being recorded, all the key presses are being appended to the text register. When it's ran, respective key presses are simulated. It is obviously also possible to store the key sequence in the register without the side effect of executing the actions yourself and then run it later.

- L q start recording macro into the register L or stop recording it if it's being recorded.
- L @ run the macro stored in L, simulating the sequence of keys being pressed.

Executing code snippets

I have created a CLUC language specification, then made a compiler and a virtual machine that can run the compiled code. This system is intertwined with this GUI project.

One of the ways to utilize the language is to simply use the built-in interpreter command line, accessible by hitting : (colon).

The effect of the [V] switch will be covered in the following text.

The commandLine on the bottom of the screen then activates, allowing the user to type code and run it by pressing enter. Pressing shift+enter creates a new line and doesn't send the code for execution yet. Once the code is executed, the return value, if any applicable, gets displayed in the text box. By default, the GUI attempts to display this value.

It's possible to use this feature as a calculator. Typing 5+2*5 and hitting enter prints out 15.

It's also useful for displaying the current values of properties of the GUI elements. For instance:
CGE.getValue()
prints the value of the currently focused GUI element on the bottom of the status window.

This default display of the result of the calculation can be overridden by appending a semicolon to the end of the line – a feature known from Matlab.

Using the commandLine, functionality not accessible from keyboard shortcuts can be accessed. Apart from all the functionality listed in the separate language documentation, it is also possible to access the different properties of the GUI elements available.

The syntax for retrieving a certain property of a GUI element is UNIQUE_NAME.getProperty()

where UNIQUE_NAME is the unique name of the GUI element and "Property" is subsituted by the name of this property, the first letter being capitalised; for instance, use

TAB1_VOLTAGE_SLIDER1.getValue() to access the value of the TAB1_VOLTAGE_SLIDER1 GUI element.

Similarly, it is possible to set different properthies of the GUI element. TAB1_VOLTAGE_SLIDER1.setValue(50) would set the value of TAB1_VOLTAGE_SLIDER1 to 50.

A special component with the name CGE exists. CGE stands for Current GUI Element, and simply means the currently focused GUI element.

CGE gets very powerful when combined with the [V] flag.

If the [V] is unset:

CGE simply means the GUI element currently focused on the current tab.

CGE.setValue(50) therefore sets the value 50 to the currently focused GUI element.

If the [V] is set:

The command will be executed multiple times, once for each selected GUI element. Each time the GUI element in question will get focused. On the first run, the first component will get foucused, on the second the second, etc. As a side effect, the "CGE" variable will change, to always contain the currently focused element. This is an extremely powerful feature. For instance, it allows the user to carry out certain actions based on given criteria. Say we wanted to hide all GUI elements which are marked with the tag "a" AND ALSO tag "b". The course of actions to achieve this is simple:

va (selects all) V (the next command should be applied to the whole selection) : (colon command)

And then we'd type in this following simple code:

```
IF(CGE.hasTag("a") && CGE.hasTag("b"))
        CGE.setVisible(0);
ENDIF
```

Instead of manipulating the visibility, we may choose to toggle the selection of different components or do virtually anything else.

Specific GUI elements

Following GUI elements exist. For each one I will list the properties they have on top of the common ones, listed in the chapter "GUI elements" and the actions that can be executed once the element is focused and "a" is pressed, additional to the actions listed in "Accessing the element" When converted to uppercase, the names listed here the so called "generic names" of the GUI components.

Slider

Image 17- Slider GUI element

A GUI element intended for setting or reading a certain value, whenever the absolute value is irrelevant, or the units are so arbitrary or hardly imaginable, there is no use knowing them (fan speed, brightness…).

Actions

- N h decrease the current value by N
- N l increase the current value by N

Properthies

(no special properties)

Display

A simple textual display used to display a measured value, intended to be read only.

Image 18- display GUI element

Actions

(no special actions)

Properties

(no special properties)

Statistical Display

A read-only GUI element, able to do statistical processing on the incoming data.

Image 19- statistical display GUI element

Whenever the Value property of this element is changed, the Statistical display waits for a user-adjustable period, which we call "delay" and then starts using this value for statistical calculations.

After an additional period, this value is discarded, is freed from the memory and will no longer be used for the calculations. This period, for which the set Value ("sample") should be used in upcoming calculations, is referred to as window width.

A sample is said to be "in the window", whenever it is being used for statistical calculations.

User can also decide, whether delay and window width should be input in seconds, or in samples, that is, in the number of calls to the setValue() function before the value gets registered and before it gets discarded.

Video, demonstrating the Statistical display GUI element:

https://youtu.be/6va4i3vuDLw

When the Value property is read from the user code, the number returned is the result of the statistical processing of all values in the window.

This also implies, that when writing some number to the Value property, subsequent read

of this property will return number inconsistent with the one that has just been written to it.

Since the calculations are computed when a new sample is added (that is, when the Value property is written to), it should be ensured by the user that the samples are being added often enough. This is simple to ensure, for instance using a Timer GUI element.

Supported statistics

The type of calculation, performed on the data in window, depends on the current calculation type set on the statistical display element. This type can be changed by the user at any time. What follows is the list of supported statistical calculations:

- Raw value
  - returns the last value written to this element; can be used in conjunction with the delay setting; Then it simply copies the incoming data to the output, delaying it.
- Average
- Sum
- Min
- Max
- Divergence
- Standard deviation
- Average of squares
- Sum of squares

Actions

- N j
  - traverse the list of different calculation types by N steps down (change type of statistical calculation)
- N k
  - traverse the list of different calculation types by N steps up (change type of statistical calculation)

Properties

- float Delay
- float WindowWidth

(Both of those properties have been described earlier in this section.)

Numeric UP/DOWN

An element for precise adjustment of a value.

Image 20 - Numeric UP/DOWN GUI element

One of the digits can be focused. Then the value of this digit can be adjusted.

When the digit is 9 and it should be added 1 to, it resets to zero and one digit to the left increases, as expectable. Digits that are not currently displayed can also be adjusted.

Whenever a digit, which is not currently displayed, is selected, user is informed about this by a small text, indicating which digit (which decade) they are about to adjust.

Image 21 - picking a digit different from the currently displayed ones

When the user then adjusts this digit to a non-zero value, it automatically appears.

Image 22 - Automatic appearing of a digit once adjusted

Actions

- N j decrease the current digit by N
- N k increase the current digit by N
- N h focus the digit to the right of the currently focused digit
- N focus the digit to the left of the currently focused digit

Properties

(no special properties)

Chart

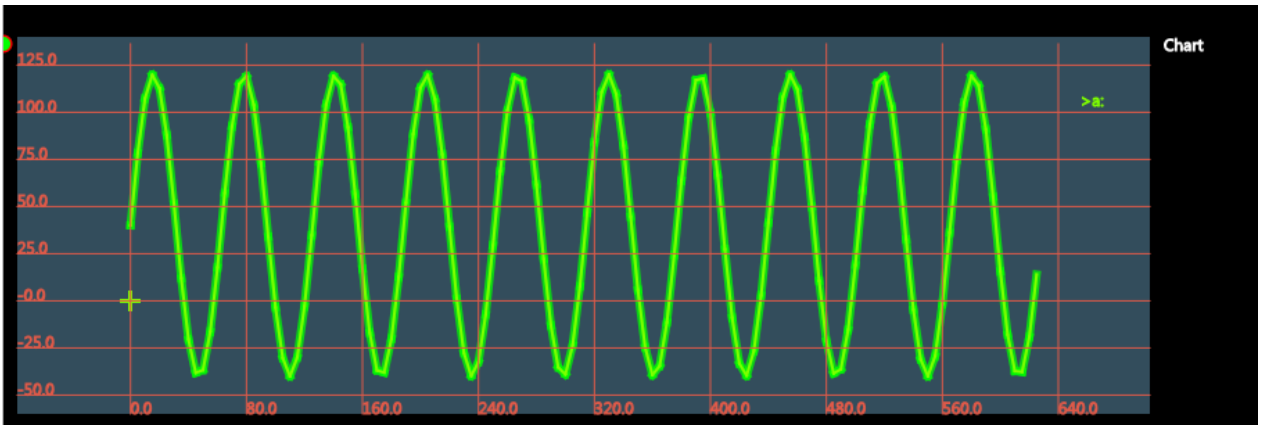


Image 23 - Chart GUI element

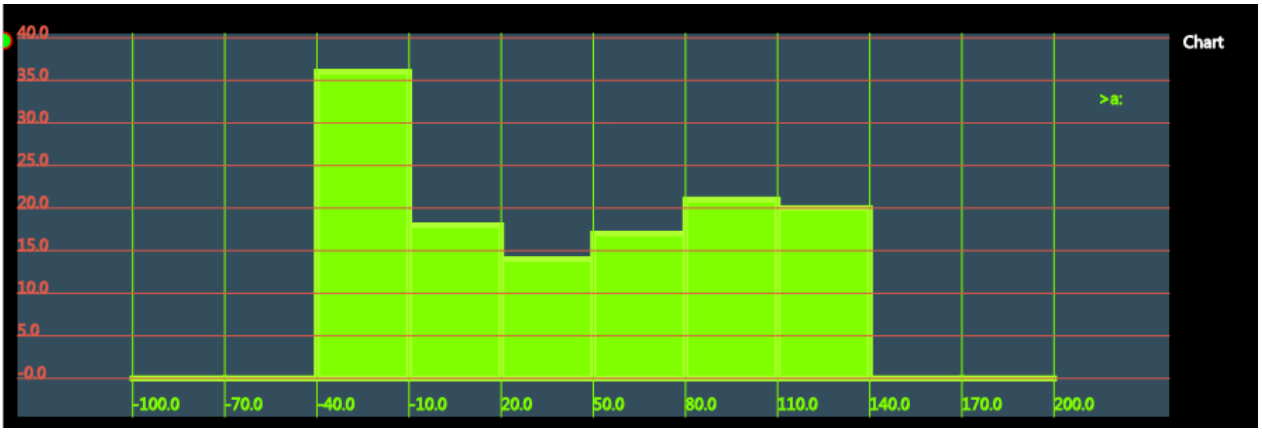


Image 24 - Histogram mode

An advanced GUI element supporting data plotting, collection and analysis. It has two modes: regular and a histogram. Those modes can be switched between at any time. This only concerns the type of display; no collected data is altered by switching the display mode.

Video, demonstrating the Chart GUI element:

https://youtu.be/ZEgEn2-JCng

In the regular mode, property/property relationships are displayed. In the histogram mode, the value on the Y axis is plotted vs. the number of times it was collected. Because the Y value is a real number, when we choose to ignore the finite resolution of floats, it is not very probable for the exact same value to be collected multiple times. Thus it is possible for the user to select amount of so called "bins" the histogram

will have. Each bin will represent a range of values instead of just one value. The histogram view is a bargraph, but is a representation of the underlying data, which can be expressed as a series of line graphs. Therefore, we will refer to the pile of data points, expressing a property/property relationship collected in one go as "line".

One chart can have numerous lines displayed in it.

When in histogram mode, the opacity of individual histograms is adjusted and multiple grids can be turned on, allowing the user to view all the histograms at the same time:

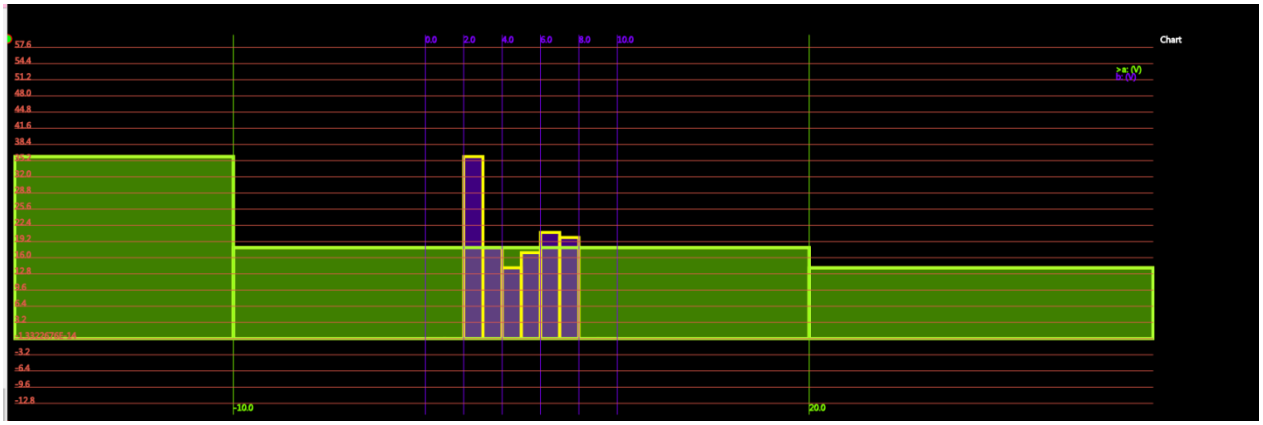


Image 25 - Histogram mode with multiple histograms displayed

Each line has one and only one unique letter assigned to it. A "current line" can be focused to be one of those lines. There are operations which implicitly affect the currently focused line. If the chart only has one line, this line is the focused one. Additionally, multiple lines can be selected and operations can be done with them. Each line can also be hidden. When it is hidden, it can still be altered by incoming data. However, it automatically gets unselected. Hidden lines cannot be selected. Each line has a cursor. This cursor can be moved along the X and Y axes programmatically. The chart has a sampling period setting. The chart can be in a "recording" state. When it is in this state, it is automatically taking samples every sampling period, by placing a datapoint at the current cursor position for all currently relevant lines. Right before doing that, it calls a special user function. User code can be placed in this function, which sets the X and Y cursor positions for the relevant lines. This code usually sets the X position to the RunTime property of the chart, that is, to the time length, for which the chart has currently been recording, and it sets the Y position to the measured value we want to plot. This plots this value against time. However, this doesn't have to be the case. Any 2 expressions can be plotted against one another.

Actions

- L ' set the "current line" to be the line with the letter L.
- L l [L2] find line that corresponds to the letter L. Then change its letter to L2 (rename). If no L was provided, change the letter for the current line.
- L v toggle selection on the line with the letter L. Toggle it on the current line if no L was provided.

- [V]L rr start/stop recording the focused line if no letter was provided, or the line with letter L, or all selected if the [V] flag was enabled.
- rs start/stop recording all lines
- L [V] f hide/show (toggle filtering) the line with the letter L. hide/show the current line if no L was provided. In case the [V] flag is set, ignore the L and instead hide all selected.
- N j set the "current line" to be the line with the letter that follows the letter of the current line alphabetically AND has 1 or more samples stored on it AND is visible. If N is selected, do it N times. (traverse down)
- N k set the "current line" to be the line with the letter that precedes the letter of the current line alphabetically AND has 1 or more samples stored on it AND is visible. If N is selected, do it N times. (traverse up)
- N J same as j, but without the condition that the line needs to have at least one sample stored or needs to be visible.
- N K same as k, but without the condition that the line needs to have at least one sample stored or needs to be visible.
- L [V] s force sample on the line with the letter L (current if no L provided) or ignore L and sample on all selected (if [V] is set). Will sample even when nothing is being recorded currently.
- L [V] dd delete current line, storing its data into register L.
- L [V] yy same as dd, but copy, leaving the line at its place.
- L p paste from register L or into the currently focused line if no L was provided, discarding the current data.
- L dl [L2] delete the line with letter L2, storing its data into register L.
- L yl same as dl, but copy, leaving the line at its place.
- h switch between the regular and histogram view.

- [V] aa autoscale both axes to fit
- [V] ax autoscale the x axis to fit
- [V]ay autoscale the y axis to fit

  When the vFlag is on, the last 3 commands will autoscale the view so that all selected lines fit in. The autoscale is also context sensitive to only be applied to the currently visible portion of the line. For instance, when autoscaling the “y” axis, a local maximum in y between the minimum “x” and maximum “x” coordinate visible in the view is found. The chart is then scaled in such a way, so that this local maximum fits in the view.

- at toggle automatic scaling to fit when new data is received

- m enter move mode; move mode can be left by hitting "escape" once in move mode, following 8 shortcuts work:

- N h move displayed area left by N steps
- N l move displayed area right by N steps

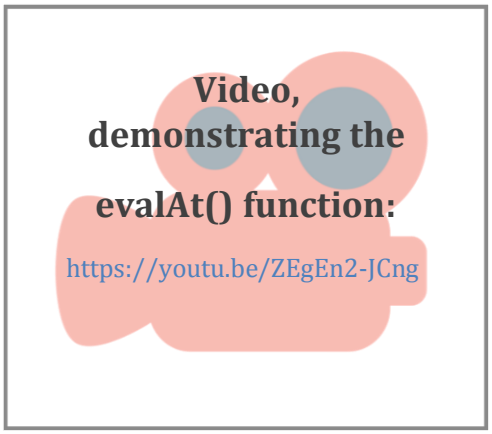
- N j move displayed area down by N steps
- N k move displayed area up by N steps
- N d increase scale of the displayed area on the x axis by N steps
- N a decrease scale of the displayed area on the x axis by N steps
- N w increase scale of the displayed area on the y axis by N steps
- N s decrease scale of the displayed area on the y axis by N steps

Functions

Following functions can be called on the chart. The syntax is [UNIQUENAME].functionName(parameters);, for instance

```
CHART1.sampleLine("a");
```

- byte sample()
  - adds a data point for each selected line if vFlag is on, or for the focused line if not on the line cursor positions
- byte sampleLine(string lineLetter);
  - adds a data point to the line with the letter lineLetter, or for the currently focused line if the lineLetter was "%".
- float evalAt(string lineLetter, float x);
  - if there is a data point stored in the line specified by the lineLetter with its x equal to the provided x coordinate, this function returns its "y" coordinate. If the provided x coordinate is placed between 2 data points, it returns some y coordinate, which is a linear interpolation between the y coordinates of those 2 points. Otherwise, returns 0. This function can be used to perform calculations using provided characteristic curves and more.



Properties

Following properties of the Chart can be accessed programmatically.

- float SampleRate
  - sampling period in seconds
- int AutoScaleMode

  - When set to 0, the chart does not autoscale when new data is present. When set to 3, it autoscales both X and Y axis. Other values are reserved for later implementation; 1 should be “only scale along X on new data”, 2 “only scale for Y on new data”, 4 “don’t scale, just move”, 5 will be “once data point is placed outside the view on the right, shift the view in such a way that it is now on the very left of the view (effectively clearing the screen once full).”
- float RunTime
  - if the chart is currently recording, this is the length in seconds it's been recording for in this current run; otherwise a 0. This property can be used to automatically collect data in time.
- int IsRecording
  - whether or not a recording is currently being taken
- int HistogramMode
  - whether or not the current view is set to histogram (1) or not (0).
- string currentLine
  - the letter associated with the line currently focused
- string name
  - the name of the chart

In addition to these, there are some properthies that are specific for each line. Acessing these from the user code has the following form:

CHARTNAME.getLineProperty("letter_of_the_line", value);

CHARTNAME.setLineProperty("letter_of_the_line", value);

for instance:

```
POWER_SUPPLY_CHART1.setLineVisible("a", 0);
```

to hide the line "a" of POWER_SUPPLY_GRAPH1.

Alternatively, “%” can be provided as a line letter, meaning “currently focused line”.

- float X the X position of the cursor that draws this line
- float Y the Y position of the cursor that draws this line
- float Visible whether or not the line is currently visible or hidden in the chart.
- float Color the color of the line; format is consistent with all other colors
- float Recorded whether or not the line is currently being recorded
- float Selected whether or not is the line selected
- float Width the width of the line

- float Focused whether or not is the line focused
- float ShowLine whether the connecting lines between the sampled points should be drawn
- float ShowPoints whether the sampled points should be signified with a circle

For technical reasons that aren't the scope of this document, the type of all line properthies was set to a float.

Callbacks

When the chart is recording, before it places points into the relevant lines, it calls a user function. This user function has the following naming convention:
[UNIQUENAME]_Sample()
where [UNIQUENAME] is the unique name of the Chart; for instance:
CHART1_Sample().

An example of a user code, which would record the value of SLIDER1 in time into the currently focused line of CHART1 would be:

```
byte CHART1_Sample()
        CHART1.setLineX("%", CHART1.getRunTime());
        CHART1.setLineY("%",SLIDER1.getValue());
ENDFUNCTION
```

This way it's also possible to record multiple lines at the same time.

Timer

Image 26 - Timer GUI element

Timers are a special GUI element.

Main purpose of timers is to automatically call user code.

Actions

- s
  - enable timer if it's not running, disable (stop) it, if it is running

Properties

- float RunTime
  - Time for which the timer has been running in seconds; -1 if not running at all;

Value of this property gets displayed on the timer body, when it is running:

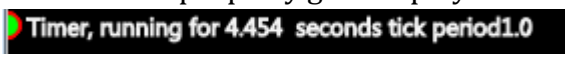


Image 27 - Running Timer

- float Enabled;
  - Whether the timer is running. Setting it to 0 makes it disabled (stops it);
- float Period
  - The time period with which the timer should execute some user code.

Once a timer is created, it is disabled by default. Enabling it will ensure that a callback function with a certain name gets called periodically, the period being 1 second by default; The timer callback function first gets called once 1 period passes; it doesn't get called immediately once the timer is enabled for the first time.

Callbacks

[UNIQUENAME]_Tick()

This function gets called once every timer period.

PID controller

Image 28 - PID controller GUI element

A simple GUI component, which serves as a discrete PID regulator.

Setting the Value property of this component sets the input error. Reading from the Value property is the way to obtain the output of the regulator.

The output is computed as (the difference between last 2 samples)*(difference constant)

+(sum of all samples)*(summing constant)+(last sample)*(proportional constant).

Remark: this algorithm is by no means perfect and should be revisited. The derivative term should be filtered out to be less noisy. However, tests have shown that it is possible to create a working regulator this way.

Actions

(no special actions)

Properties

- float P
  - the proportional constant
- float I
  - the integral (summing) constant
- float D
  - the derivative (difference) constant

Callbacks

(no special callbacks)

Global functions

It is possible to run any code snippet by pressing ":" and typing in the code before hitting enter. Apart from changing properties of the GUI elements present and calling and defining user functions and declaring variables, it is also possible to run global functions. Using them the user can achieve things that otherwise wouldn't be possible.

Math functions

There should be some math functions here, but... This is easy to implement, since those will mostly be direct calls to Java functions and I really don't feel like describing which one of the 20+ functions does what and I don't really feel like implementing it either. Let's leave this for some really late development stage.

Print functions

2 print functions are available, which print their output into the status window of the GUI.

- byte printText(string text) print the text given
- byte printNumber(float number) print the number given

  Both functions return 0 in all cases.

Map functions

See "Key Mappings" for more information about what mapping is.

Apart from the instanceMap and typeMap functions, mentioned elsewhere, there is one function to add global mappings, that are active when no elements are being edited:

map(string sourceSequence, string targetSequence)

Running this command will ensure that the GUI will thing the sequence "targetSequence" was pressed whenever "sourceSequence" was actually pressed, provided context is met. code() opens the user code in the built-in editor, compiling it once it closes

Tabs

- int tabnew(string name) Creates a new tab with the given name.

-int tabclose(string name)

Close the current tab.

Mischellaneous

- int sendGlobalKeys(string simulatedPresses)

simulates typing the string simulatedPresses, running actions corresponding to those keystrokes

- int new[genericName](string name)
  - Set of functions, such as newDisplay(name) or newSlider(name). They create a new GUI element, leaving its settings on default; (See "Specific GUI elements" for the list of names).
- int registers() Prints out the list of registers and their content in the textbox. Returns 0.

CLUC language

Remark: This section is only intended as an overview of the CLUC language. Detailed information, regarding the way this language is compiled, can be found in a separate document (see List of attachments:).

CLUC stands for C-like Language for User Code. It is an imperative, statically-typed computer programming language, supporting structured programming, lexical variable scope and recursion.

The C-like notion refers mostly to the semantics of this language, the syntax being slightly different.

This language is compiled into a so-called bytecode: a representation, which is easier to interpret by a computer. It is then interpreted from this intermediate representation using a purpose-built virtual machine.

This virtual machine is written in Java by me. I also wrote another virtual machine, able to interpret the compiled language, in C. I am hereby restating the previously stated to avoid confusion: The virtual machine, which interprets my CLUC language, is written by me, and is different from the JVM.

Why make my own language?

Although I could try to come up with reasons, why using my language is better than using an already-made one, the truth is, the reasons are mostly historical.

1) Although the well-structured nature of this document may give the false impression the scope of this project has always been clear, opposite is the case. I did not originally plan to introduce any sort of automation and mostly just wanted a simple parser, which I would then use to perform calculations using simple expressions. Introducing "full-blown" languages, such as Python, seemed like overkill at the time.

2) I have also originally intended the overall architecture to be different. Instead of having a PC and child modules, I wanted the whole platform to be standalone.

   I wanted it to consist of the child modules and a main module, which could optionally be connected to a PC. The main module would provide all the functionality currently provided by the GUI on the PC. However, I wanted to base the main module around the ESP8266 development board. This board is extremely inexpensive; however, it doesn't run any OS in the classical sense of word. In layman's terms, it's more like an Arduino than like Raspberry Pi.

   In order for this main module to be able to do real-time calculations on the collected data, I figured I need to be able to run some interpreted language for it. At the time I was contemplating this, only limited support for Lua and Python existed, but both had their flaws and were very resource-heavy. Therefore, I wrote a simple virtual machine, which could even be run by an Arduino Uno, interpreting the code written by me and compiled by my compiler.

   Although I have successfully tested this virtual machine, I have realized that a much more sane approach would be to use a device which runs Linux for the main module, and will do so in the future.

4) I have managed to get my hands on an old Atari computer. Thus I became very interested in the inner workings of this beautiful machine, which got me interested into compilers and virtual machines. I partially decided to write this language as a fun challenge.

Data types

The CLUC programming language is statically-typed. This means that all variables must first be declared before they can be used. [3] This involves stating the variable's type and name:

```
int chicken;
chicken=1;
```

Doing so tells the compiler that a field named "chicken" exists and holds numerical data, value of which is "1". A variable's data type determines the values it may contain, plus the operations that may be performed on it. In addition to int, the CLUC language supports two other primitive data types.

The supported data types are following:

- `byte`
    - number ranging from 0 to 255 (inclusive), 8 bits
- `int`
    - number ranging from -32768 to 32767 (inclusive), 16 bits
- `float`
    - single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion

Remark: CLUC also has some support for Strings. These are stored in a part of memory, called heap as a sequence of byte numbers, where each number corresponds to one letter of the string, encoded in ASCII. Zero (ASCII NULL, byte of value zero) is placed on the end of this string. The address on which the string is stored can be passed in a variable of type `int`.

Literals

Literals are simply a notation for representing a fixed value. [wiki]

For instance, in the code

---

[3] This definition, and the following definitions of the data types are partially inspired by (but not always strictly identical to) the respective parts of:
The Java Tutorial: A Short Course on the Basics
By Raymond Gallardo, Scott Hommel, Sowmya Kannan, Joni Gordon, Sharon Biocca Zakhour

```
slepice=3+2;
```

3 and 2 are literals.

There are 4 different types of literals in the CLUC language:

- byte literal
- int literal
- float literal
- string literal

The first 3 types of literals mentioned above are numeric literals.

When a numeric literal is encountered in the code, its type (e.g. whether it's a byte literal, an integer literal or something else) is determined based on the magnitude of this number. For instance, 255 will be treated as a byte literal, while 256, which exceeds the maximum number, which can be expressed in a byte, will be treated as an integer literal.

When a floating point is present in the number literal, it will automatically be treated as a float literal.

A dot (".") is used as a floating point.

Additionally, a scientific notation can be used in numeric literals. For instance:

```
a=2+1e3;
```

sets a to 1002.

At this stage, when introducing negative literals, the user has to place 0 before them, as such:

```
a=0-3;
```

as opposed to

```
a=-3;
```

This behavior has only been discovered recently and is considered a bug; it is scheduled to be fixed ASAP.

Operators

Operator is a symbol, used to perform operation.[4]

Most operators are used in CLUC the way they are used in algebra and common programming languages, such as C.

For instance,

```
a=3+2;
```

sets the value of a to 5.

Operators are evaluated left to right by default. For instance in the following code:

```
a=2+3+4;
```

"2+3" is evaluated first to be 5 and then 4 is added to this intermediate result.

There are 2 exceptions to this rule.

Operators with higher precedence get evaluated first. For instance,

```
a=5+2*5;
```

Evaluates to 15, as opposed to 35, since multiplying is of higher priority than addition.

Order in which the parts of the expression get evaluated can be forced using parentheses.

For instance,

```
a=(5+2)*5;
```

---

[4] This definition is slightly inspired by the one on:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

evaluates to 35.

Formally, we may wish to say that evaluating an expression, which consists of 2 expressions and an operator between them, simply means evaluating those 2 expressions, replacing them with a literal, performing a calculation determined by the type of the operator on those literals and then replacing the whole string (the 2 expressions and the operator between them) with a literal, denoting the result of this calculation.

This recursive definition will be formally sound when we introduce the formal definition for an expression and for "evaluating a function call" later on.

CLUC has several binary operators, that is, operators that take 2 input arguments.

The priority of those operators and their meaning is consistent with algebra.

Algebraic operators

Following algebraic operators are present: **+**, **-**, ***,/**

The return type of such operators depends on the type of the operands; For instance, if two byte operands are added together, the result is also a byte. It's also possible to combine operands of a different type. For instance, one can add a float to a byte. In such a case, the "shorter" (byte) operand will be upcast (turned into) the "longer" type (float). Then, the operation will be performed.

Comparison operators

Operators, return value of which is either 1 or 0, based on the operands. For instance, the expression "a>b" evaluates to 1 if and only if the variable "a" is greater than the variable "b" and it evaluates to 0 otherwise.

Following algebraic operators are present: **>**, **<**, **>=**, **<=**, **==**

Logical operators

Operators, return value of which is either 1 or 0, based on the operands. The type of operands has to be a byte.

Following logical operators are present: &&, ||

- &&

  AND operator; return value is 1 IFF both operands are 1; it is 0 otherwise.

- ||

  OR operator; return value is 1 IFF at least one of the operands is 1; it is 0 otherwise.

In addition to the aforementioned binary operators, the not() function is used for negation. This was done in order to circumvent the lack of a native CLUC language support for unary operators. This function returns a byte; the value of it is 1 if its only (byte) argument is 0 and 0 otherwise.

Function calls

Function call has the form of name of the function, which is being called, left parenthesis "(", expressions (arguments) separated by commas and right parenthesis ")". If the function, called in this function call, doesn't accept any arguments, the call has the form of a function name, followed by a left parenthesis, followed immediately by a right parenthesis:

```
doSomething()
```

Functions calls can, among other things, be used to calculate some value, based on the arguments provided. When a function call is said to be evaluated, we simply mean replacing the call by the literal value, calculated by the respective function. Such value is said to be returned from this function. For instance:

```
sqrt(9)
```

evaluates to 3, since sqrt is a function used to calculate the square root of the input argument – or, in other words, sqrt returns the square root of its argument.

All functions return a value.

Evaluating a function may also have additional effects. For instance, calls to the built-in function printText result in text being written out in the status window.

Such functions usually always return 0, as this additional effect is the primary reason behind using this function.

There are 2 main types of a function call in CLUC.

First one is a function call to a built-in function.

Second one is a user function.

Syntactically, those 2 types of calls are identical. The main difference is that user function call is evaluated using CLUC code written by the user, built-in functions are evaluated using Java code, which is usually much faster.

Expressions

Expressions are a syntactically valid sequence of literals, variable names, operators and function calls.

Following list will determine, when such a sequence is said to be syntactically valid.

1) A single literal is a valid expression. Thus, "3" (without the quotation marks) is a valid expression

2) A single sequence of characters, acceptable as a variable name is a valid expression. Thus, "slepice" (without the quotation marks) is a valid expression.

3) A sequence of characters, acceptable as a function name, followed by a left parenthesis "(", valid expressions separated by commas ",", followed by a right parenthesis ")" (in other words, a valid function call) is also a valid expression.

   Thus, "kokodak(2,3,4)" is a syntactically valid expression.

4) A sequence of an expression, followed by a binary operator, followed by another expression, is also an expression.

5) A sequence of a left parenthesis, followed by an expression, followed by a right parenthesis, is also an expression.

6) if A is a string representation of a valid expression, we can form a string B by inserting a whitespace character (space or tab) on any position in A. If this operation hasn't caused changes in any substring of A, which would be acceptable as a variable name or a function name, resulting string B is also a valid expression.

7) Nothing else is a valid expression.

Please notice that this list only determines, what is a syntactically valid expression; however, strings can be created in such a way that they represent an expression, which isn't correct semantically. For instance, while

```
chicken(2,3)
```

It can be meaningless, and therefore semantically invalid, for various reasons – for instance, when no function chicken exists, or if it accepts a different number of arguments.

If the expression is both syntactically and semantically valid, it can be evaluated.

Evaluating an expression means replacing all operators with their operands with a value, calculated based on the given operator type and the operands, and replacing all function calls with their arguments by the return value of this function, based on the arguments provided.

The order in which the individual replacements will be conducted is based on both the order the parts of the expression appeared in and the operator precedence.

Statements

Statement is the smallest standalone element, which expresses some action to be carried out. [63]

Simple statements in CLUC are written on a single line, ended with a semicolon (;).

There are 3 types of simple statement in CLUC:

Variable declaration

This statement simply says that a variable of a given name and type will be introduced. It consists of the type of the variable, followed by the name. The name has to be a legal name of a variable. That is, it can contain letters and numbers only and must not begin with a letter. As all statements, this one is also ended with a semicolon.

Examples:

```
int slepice;

float kokodak;

byte kokodak123;
```

Variable assignment

This type of statement is used to assign a value to a variable. The statement consists of the name of the variable we want to assign to, followed by an equation symbol ("="), followed by an expression, which the user wishes to evaluate to a literal, which should then be used for the assignment, ended with a semicolon (;).

Examples include:

```
slepice=3+2;
kokodak=slepice+2;
kokodak123=kokodak*slepice+123+sqrt(4);
```

Function call statement

This statement consists of a single function call, ended by a semicolon (;).

The arguments of this function call are any valid expressions, including expressions formed by other function calls. This means, that the following are examples of valid function call statements:

```
doSomething(3+2);
doSomething(doSomething(3+2));
doSomethingElse(doSomething(),4+5);
```

But the following aren't:

```
doSomething(3+2)
```

(missing semicolon)

```
doSomething()+doSomething()
```

(Multiple function calls on one line, without the other function calls being parts of arguments of the first function call.)

Order of execution

When statements are placed below each other, each on its own line of code, they are executed in the order they appear in this code.

For instance, in the following code:

```
int a;
printText(„hello“);
a=3;
printNumber(a);
a=5;
printNumber(a);
```

the text “hello” will get printed out; then, variable a will be set to 3, it will get printed out, get set to 5 and get printed out again.

However, there are exceptions to this rule.

Namely, a variable declaration statement can be placed after it is used without any issues:

```
a=3;
printNumber(a);
int a;
```

There are other exceptions:

Control flow statements

Control flow statements breakup the flow of execution by employing decision making, branching and looping[java doc]. They allow the CLUC program to conditionally execute particular blocks of code.

These statements are complex statements and are not ended with a semicolon.

IF-ELSE-ENDIF statements

IF statement consist of the keyword “IF” followed by an expression in parenthesis.

ELSE statement is simply the ELSE keyword alone on its own line.

ENDIF statement is simply the ENDIF keyword alone on its own line.

IF block is a piece of code starting with an IF statement and ending with a matching ENDIF statement. Optionally, a matching ELSE statement can be present between the IF and ENDIF statements. Any valid statements can appear anywhere between the IF and ENDIF statements.

Statements that follow the IF statement up to the matching ELSE statement (if any present) or the matching ENDIF statement (if no matching ELSE statement was found) will be evaluated, in the order they appear, if and only if the expression, which was part of the IF statement in question, evaluated to a non-zero value.

Otherwise, given that a matching ELSE statement was found, statements below this ELSE statement and the matching ENDIF statement will be executed in the order they appear in.

Once the ELSE or ENDIF statement is reached, the execution continues, starting with the first line that follows after the matching ENDIF statement.

For the sake of completeness, let's define what exactly we mean by a matching ELSE and ENDIF statements to a given IF statement, with the following pseudocode:

1) set current depth to 0.
2) while you haven't reached the end of code,

look at the current line

3) If it's an IF statement, add 1 to the current depth. Then give this IF statement a number, equal to the current depth.
4) If it's an ELSE statement, just give it a number, equal to the current depth.
5) If it's an ENDIF statement, give it a number, equal to the current depth. Then subtract one to the current depth.
6) Go to next line if possible, then go to step 2).
7) If we now take any pair of 2 IF, ELSE or ENDIF statements, such that the type of those 2 statements is different (for instance, not 2 IF statements) and those 2 statements have been assigned the same number, they are said to be matching.

FOR-NEXT statements

FOR statement consist of the FOR keyword, followed by a statement, followed by an expression, followed by another statement. Those 3 are separated by semicolons, wrapped together in parenthesis.

The first statement must be a variable assignment. No restrictions apply to the following expression. The statement that follows must be a variable assignment.

NEXT statement is simply the NEXT keyword alone on its own line.

FOR block is the FOR statement, followed by an arbitrary number of other statements, ended with a NEXT statement.

One such example of a FOR block would be:

```
FOR (i=0; i<5;i=i+1)
        printNumber(i);
NEXT
```

When a FOR block is encountered, following happens:

1) When the FOR statement is evaluated, first simple statement of the FOR statement is evaluated.
2) The expression in the FOR statement (the part between the two semicolons) is evaluated. When the result of this expression 0, the program execution continues starting with the first line after the matching NEXT statement. Otherwise it continues executing, starting with the first line after the FOR statement.
3) Once the NEXT statements is reached (evaluated), the last expression (after the second semicolon) in the FOR statement is evaluated. Then, the program execution continues, starting from the matching FOR statement. Thus, the next evaluated line of code will be the matching FOR statement.

Thus, running the code in our last code example would give us following printout:

```
0
1
2
3
4
```

User function call is another exception from the regular program flow.

Whenever a function call is being evaluated, the expressions present are evaluated “inside out”. For instance, in the following example:

```
printNumber(sqrt(4));
```

sqrt(4) is evaluated first and is replaced by the literal “2”. Then, it’s printed out by the printNumber function.

This is logical, since if the sqrt call wasn’t evaluated at the time printNumber is run, printNumber wouldn’t know what number to print.

When a call to a built-in function is evaluated, actions can happen as a side-effect and then the call to this function is replaced by the value, calculated by this function.

This value is known as the return value of the function.

In the last example, sqrt(4) gets replaced with 2 as a result of the evaluation and then printNumber(2) is evaluated. Side effect of this is printing the number 2 out. Then, printNumber(2) is replaced by 0. This 0 is then discarded, since having 0 alone on a line makes no sense.

User function is a function, side effects of which are executed by running user-defined CLUC code. The return value is also defined by this code.

Function body consists of a function definition token, followed by any statements, followed by the ENDFUNCTION token.

Optionally, RETURN tokens may be placed at any position between the function definition and ENDFUNCTION tokens.

Function definition token consist of a data type keyword, followed by the name of the function, followed by a so-called formal argument list.

This formal argument list begins with a left parenthesis, followed by pairs of data type keywords and variable names, separated by commas, and ends with a right parenthesis.

RETURN statement is simply the RETURN keyword, followed by a valid expression, ended with a semicolon.

ENDFUNCTION statement is simply the ENDFUNCTION keyword alone on a single line.

An example of a function body would be:

```
int addNumbers(int a, int b)
        RETURN a+b;
ENDFUNCTION
```

The comma separated list of expressions, which can be evaluated to literals, which follows the function name in a function call, is called actual argument list. Once all these expressions are evaluated, this list only contains literals.

When a call to a user function starts evaluating, the individual expressions in the actual argument list are evaluated to literals. Next, the program continues execution from the matching function definition. This is unambiguous, since it is not possible for multiple functions with the same name to exist. Variables mentioned in the formal argument list of this function definition get declared automatically and are assigned values from the actual argument list (which now only contains literals) in the order they appear in. That is, first variable in the formal list is assigned the first value of the actual list, second variable in the formal list is assigned the second value of the actual list, etc.

These variables can then be used in any expression in the function block. Once the function block is left (the code currently executed is not a part of this function block), those variables are deleted. This is also true for any variables explicitly declared in the function block. Such variables are called local and we say that their scope is the function block.

One a RETURN statement is reached, the expression it contains is evaluated. The function call which led to the statements in this function block to be evaluated gets replaced by the literal, gained from evaluating this expression.

Example:

Let's look at the following code and assume the function addNumbers is a user function, introduced in our last example.

```
printNumber(addNumbers(1+1,2));
```

This code will first evaluate what 1+1 equals to, then local variables a and b will be defined inside function addNumbers, they will be assigned values 2 and 2; on the RETURN statement line, these 2s will be added together, 4 being the result.

Next, the addNumber function call will be replaced by this 4:

```
printNumber(4);
```

Finally, the built-in function printNumber will be called, printing out the number 4.

When the ENDFUNCTION statement is reached, it is treated as if

RETURN 0; was reached instead.

Thus, return statements are not mandatory.

Conclusion

Remark:

The conclusion would usually be expected to include ideas for further development of the project.

For the sake of brevity of the conclusion section, I have decided to move such content to a separate document (see the List of attachments:).

Long-term goal of my instrumentation platform project, which I will continue for my Diploma thesis, is to build a system, able to measure or generate quantities in real time, perform real-time calculations on them, plot the results of those calculations, and provide some means of automation. This platform should consist of a main module and child modules. The main module should handle the processing and user interface, while the child modules should handle the data collection and controlling outputs, if any.

I have made a proof-of-concept child module, which I could use to test the functionality of the data processing/collection software I have written.

I have specified the communication protocol which should be used between the main module and the child modules and have mostly implemented it (it is functional, but doesn't have all the features I'd like it to have).

I have written software in Java, which:

- Can automatically ask the connected module, what sort of GUI should be provided for it and then display this GUI (see Default GUI)
- Allows the user to then modify said GUI by reorganizing the elements, changing their appearance and functionality and also by placing additional user elements on any position. (see Moving the elements)
- Has a very powerful mechanism, able to assist the user, when changing the properties of multiple GUI elements at once, or when moving multiple elements around. (see Filtering, Selection)
- Can collect data from the module in real time, displaying it for the user to see
- Can perform calculations, using this data in real time, using mathematical expressions, as well as using characteristic curves defined by their datapoints; those 2 approaches can be freely combined and said curves can be imported from spreadsheet editors or Matlab and also exported to them. (see Executing code snippets, Chart Functions)
- Can perform statistical analysis on collected or computed data in real time; statistical functions include minimum, maximum, average and standard deviation; they operate in a floating window, width of which is adjustable; user-adjustable delay can be used to hold the samples before they enter the floating window. (see Statistical Display)
- Can plot the collected or calculated data (including the data, which is the result of the aforementioned statistical analysis) in real time. Multiple values can be plotted simultaneously. It is also possible to plot values one by one. One chart can have multiple plot lines stored in it. (see Chart)
- Can export this plotted data to spreadsheet editors or imported from them
- Can display histograms and update them in real time, as the data is being plotted
- Can handle automation, such as directing the outputs of connected modules to change their value in some way, whenever a condition, such as the measured inputs being in a certain range, is met; (see Callbacks and user code). It's also possible to import a curve from other software and then set outputs of some modules to follow this curve; (see Chart) it's also possible to automatically execute code snippets once every user-adjustable period (see Timer)

- Provides a simple PID regulator with user-adjustable constants; this regulator can be used, among other things, to control the outputs of some module based on the inputs of this or other module. (see PID controller)

I have then covered this functionality in this document. I have also directed and made videos, showcasing some of the functionality listed. (see List of attachments:)

I have conducted a brief research, regarding products similar to what my Instrumentation platform project will be once complete and have presented the results of this research in this paper, in the section Similar projects.

All the GUI elements, as listed in the chapter "Specific GUI elements," were created by me, in that the code used to display them, as well as the code used to operate them is mostly mine, and only uses the Canvas [8]element of the JavaFX [6]library.

For example, the code to plot the charts, code to automatically determine the correct ticks size, autoscaling routines or the histogram drawing routine were written by me.

I have only used default JavaFX routines to draw rectangles, lines, text and similar simple shapes, to set the color of those objects, etc.

I have made a language specification for my language called CLUC (C-like Language for User Code) and have written a bytecode compiler and interpreter for this language. Both the compiler and interpreter are written in Java. They are intertwined with the GUI application. This language is used to perform calculations on the incoming data, as well as for automation.

Additionally, I have written a (now abandoned) proof-of concept interpreter of CLUC in C for Arduino.

I have briefly covered the aspects of this language in this document. I have also written an additional document, which covers the inner workings of the compiler.

I have written so-called unit tests to test the behavior of the compiler and the interpreter.

I have written a Javadoc (language-specific documentation) for the compiler, interpreter and the GUI program.

I have hoped to create some hardware for the individual modules, other than the proof-of-concept one; this has been strongly suggested against by my supervisor, and I wholeheartedly agreed, as the scope of my work is rather extensive as it is.

Other than that, I am very happy with the work I have done. It will lay the groundwork for my diploma thesis, and once hardware modules are made, I truly believe it to become a priceless tool for electronics hobbyists and engineers worldwide.

Additionally, this work is also as a case study of heavily keyboard oriented, Vim inspired user interfaces, and as such could be used as a source of ideas when developing user interfaces, based on this popular text editor.

Sources

[1] "Vim, the editor," [Online]. Available: http://www.vim.org/about.php. [Accessed 19 5 2016].

[2] "Ranger," [Online]. Available: http://ranger.nongnu.org/. [Accessed 19 5 2016].

[3] "NetBeans IDE," Oracle Corporation and its affiliates, 2017. [Online]. Available: https://netbeans.org/. [Accessed 19 5 2017].

[4] "jVi - vi/vim editor clone - plugin detail," [Online]. Available: http://plugins.netbeans.org/plugin/2802/jvi-vi-vim-editor-clone. [Accessed 19 5 2017].

[5] "Java™ Platform, Standard Edition 8 API Specification," Oracle and/or its affiliates, [Online]. Available: http://docs.oracle.com/javase/8/docs/api/. [Accessed 19 5 2017].

[6] "Java Platform, Standard Edition (Java SE) 8," Oracle and/or its affiliates, 2016. [Online]. Available: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm. [Accessed 19 5 2017].

[7] T. Jarvi, "rxtx - a Java cross platform wrapper library for the serial port," [Online]. Available: https://github.com/rxtx/rxtx. [Accessed 19 5 2017].

[8] "Class Canvas," Oracle and/or its affiliates., 2008, 2015. [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html. [Accessed 19 5 2017].

[9] S. Hommel, "Working with Canvas," Oracle, April 2013. [Online]. Available: http://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm. [Accessed 19 5 2017].

[10] Arduino, "Arduino Duemillanove," 2017. [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardDuemilanove. [Accessed 19 5 2017].

[11] Arduino, "Access the Online IDE," [Online]. Available: https://www.arduino.cc/en/main/software. [Accessed 19 5 2017].

[12] "GNU Bison," Free Software Foundation, 8 6 2014. [Online]. Available: https://www.gnu.org/software/bison/. [Accessed 19 5 2017].

[13] S. C. Johnson, "Yacc: Yet Another Compiler-Compiler," AT&T Bell Laboratories,

[Online]. Available: http://dinosaur.compilertools.net/yacc/. [Accessed 19 5 2017].

[14] J. Gruber, "Daring Fireball," 17 December 2004. [Online]. Available: https://daringfireball.net/projects/markdown/. [Accessed 19 5 2017].

[15] "Pandoc - a universal document converter," [Online]. Available: http://pandoc.org/. [Accessed 19 5 2017].

[16] Microsoft, "Microsoft Word—the world's most popular word processing program," Microsoft, 2017. [Online]. Available: https://products.office.com/en-us/microsoft-word-2010. [Accessed 19 5 2017].

[17] "TME - výsledky hledání výrazu "multimetr"," TME, [Online]. Available: http://www.tme.eu/cz/katalog/#search=multimetr&s_field=accuracy&s_order=DESC&id_category=100164&page=1. [Accessed 19 5 2017].

[18] "FLUKE FLUKE 88X-USB," [Online]. Available: http://www.tme.eu/cz/details/flk-884x-usb/prislusenstvi-ostatni/fluke/fluke-88x-usb/. [Accessed 19 5 2017].

[19] "BEHA-AMPROBE 38SW," [Online]. Available: http://www.tme.eu/cz/details/38sw-a/meridla-software/beha-amprobe/38sw/. [Accessed 19 5 2017].

[20] xaizek, "Some notes of computer stuff," 13 8 2016. [Online]. Available: https://xaizek.github.io/2016-08-13/big-list-of-vim-like-software/. [Accessed 17 5 2017].

[21] "Orange PI," [Online]. Available: http://www.orangepi.org/. [Accessed 19 5 2017].

[22] Microchip Technology Inc., "Two Channel Analog Front End," 2009. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/22192a.pdf. [Accessed 17 5 2017].

[23] "TechTip: Choosing a Software Environment for Data Acquisition," Measurment computing tm., 2017. [Online]. Available: https://www.mccdaq.com/TechTips/TechTip-3.aspx. [Accessed 19 5 2017].

[24] "LabVIEW - From Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/LabVIEW. [Accessed 19 5 2017].

[25] National instruments, "LabVIEW System Design Software," [Online]. Available: http://www.ni.com/labview/. [Accessed 12 4 2017].

[26] "LabView," [Online]. Available: https://images-blogger-opensocial.googleusercontent.com/gadgets/proxy?url=http%3A%2F%2F2.bp.blogspot.com%2F-qDGfgkDIOvE%2FUTLuo3_Gm-I%2FAAAAAAAAAFM%2FMdGXqyy5hDs%2Fs1600%2Fclip_image002_2011061711

3353.png&container=blogger&gadget=a&rewriteMime=image%2F*.

[27] Digilent, "LabVIEW Home Bundle," [Online]. Available: http://store.digilentinc.com/labview-home-bundle/. [Accessed 1 5 2017].

[28] ISES, "Školní experimentální systém ISES," [Online]. Available: http://www.ises.info/old-site/. [Accessed 12 11 2016].

[29] "Katalog učebních pomůcek pro fyziku, chemii a biologii," [Online]. Available: http://www.ises.info/old-site/fotky/obr05.jpg. [Accessed 19 5 2017].

[30] "ISES - moduly," 23 7 2014. [Online]. Available: http://www.ises.info/old-site/index.php?f=moduly. [Accessed 17 5 2017].

[31] Internetové Školní Experimentální Studio, "Relé," [Online]. Available: http://www.ises.info/old-site/index.php?s=m&f=m16.

[32] Hadex, "Relé RP210 12V/2P," [Online]. Available: http://www.hadex.cz/l629-rele-rp210-12v2p/. [Accessed 19 5 2017].

[33] "Internetové Školní Experimentální Studio (obrázek relé)," [Online]. Available: http://www.ises.info/old-site/moduly/pic/M_rele.jpg.

[34] Vernier, "Logger lite," [Online]. Available: https://www.vernier.com/products/software/logger-lite/. [Accessed 12 11 2016].

[35] Vernier Software & Technology, LLC., "Logger Pro®," [Online]. Available: https://www.vernier.com/products/software/lp/. [Accessed 19 5 2017].

[36] Vernier Software & Technology, LLC., "Logger Pro," [Online]. Available: https://www.vernier.com/images/cache/screenshot.lp._graphmatch.001.443.332.png. [Accessed 19 5 2017].

[37] Vernier, "LabQuest 2 for Data Collection and Analysis," [Online]. Available: https://www.vernier.com/platforms/labquest-2/. [Accessed 12 11 2016].

[38] Vernier Software & Technology, LLC., 2017. [Online]. Available: https://www.vernier.com/images/masters/banner.labq2.image-only.001.jpg. [Accessed 19 5 2017].

[39] Vernier, "Vernier Engineering Projects with LEGO® MINDSTORMS® Education EV3," [Online]. Available: https://www.vernier.com/products/books/ep-ev3/. [Accessed 12 11 2016].

[40] Vernier, "Lab Equipment," [Online]. Available: https://www.vernier.com/products/lab-equipment/. [Accessed 12 11 2016].

[41] Extech, "Analog or Digital Triple Output DC Power Supply," [Online]. Available: http://www.extech.com/resources/382203_13_22_25data.pdf. [Accessed 19 5 2017].

[42] Vernier, "Vernier Arduino shield," [Online]. Available: https://www.vernier.com/products/interfaces/bt-ard/?search=arduin&category=autosuggest. [Accessed 12 11 2016].

[43] Pasco Scientific, "850 Universal Interface - UI-5000," [Online]. Available: https://www.pasco.com/prodCatalog/UI/UI-5000_850-universal-interface/. [Accessed 13 11 2016].

[44] Pasco Scientific, "850 Universal Interface," [Online]. Available: https://www.pasco.com/images/products/ui/UI5000_MAIN_170975.jpg. [Accessed 19 5 2017].

[45] Pasco scientific, "AirLink - PS-3200," [Online]. Available: https://www.pasco.com/prodCatalog/PS/PS-3200_airlink/index.cfm. [Accessed 13 11 2016].

[46] Pasco scientific, "SPARKvue Software," [Online]. Available: https://www.pasco.com/sparkvue/. [Accessed 13 11 2016].

[47] Pasco Scientific, "SPARK Element," [Online]. Available: https://www.pasco.com/prodCompare/spark-element/. [Accessed 19 5 2017].

[48] Pasco scientific, "PASCO Capstone Software," [Online]. Available: https://www.pasco.com/prodMulti/pasco-capstone-software/index.cfm.

[49] Pasco scientific, "Analog ELVIS Adapter," [Online]. Available: https://www.pasco.com/prodCatalog/CI/CI-6718_analog-elvis-adapter/. [Accessed 13 11 2016].

[50] Papouch store s.r.o, "TME multi: Temperature and humidity via Ethernet," [Online]. Available: https://www.papouch.com/en/shop/product/tme-multi-temperature-humidity-via-ethernet/. [Accessed 17 5 2016].

[51] Papouch store s.r.o, "TME - Ethernet Thermometer," [Online]. Available: https://www.papouch.com/en/shop/product/tme-ip-ethernet-thermometer/. [Accessed 17 5 2016].

[52] Papouch s.r.o, "IP Relay: IP relé se sériovým portem," [Online]. Available: https://www.papouch.com/cz/shop/product/iprelay-web-rele-a-rs232/. [Accessed 17 5 2016].

[53] Papouch store s.r.o, "Wix - measuring software," [Online]. Available: https://www.papouch.com/en/shop/product/wix/. [Accessed 17 5 2017].

[54] Papouch s.r.o, "Papouch," [Online]. Available: https://www.papouch.com/en/ariadne/file_generators/dbfile.php?_fileId=7340&_fileName=wix-window.png&_site=papouch2_web. [Accessed 19 5 2017].

[55] myDevices Cayenne, "Your Dashboard. Your Design.," [Online]. Available: https://mydevices.com/cayenne/features/. [Accessed 17 5 2016].

[56] myDevices Cayenne, "Supported hardware," [Online]. Available: https://mydevices.com/cayenne/supported-hardware/. [Accessed 17 5 2017].

[57] V. Kolesnik, "Cayenne dashboard simplifies IoT project building on Raspberry Pi," 14 10 2016. [Online]. Available: http://www.radiolocman.com/review/article.html?di=181891. [Accessed 19 5 2017].

[58] V. Kolesnik, "Cayenne dashboard simplifies IoT project building on Raspberry Pi," 14 10 2016. [Online]. Available: http://www.rlocman.ru/i/Image/2016/02/25/Cayenne_flexible_dashboard.jpg. [Accessed 19 5 2017].

[59] myDevices Cayenne, "Create Triggers in a Snap," [Online]. Available: https://mydevices.com/cayenne/features/triggers/. [Accessed 13 11 2016].

[60] John, "Horace goes skiing," [Online]. Available: http://horacegoesskiing.com/index.php/2016/01/26/quick-and-dirty-raspberry-pi-triggers-using-octoblu-and-cayenne/. [Accessed 19 5 2017].

[61] John, "Horace goes skying," [Online]. Available: http://horacegoesskiing.com/wp-content/uploads/2016/01/cayenne1.png. [Accessed 19 5 2017].

[62] myDevices Cayenne, "Video Tutorials," [Online]. Available: https://mydevices.com/cayenne/videos/. [Accessed 17 5 2016].

[63] "Statement (computer science) - From Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/Statement_(computer_science). [Accessed 19 5 2017].

Acknowledgements

Although I have mostly done the work described in this document myself, I still would like to thank the people who have made my life amazing, both during the time of writing and otherwhiles.

First and foremost, I would like to thank my mom and my family, my girlfriend, Petr Brož, and all the other people who have always been very supportive when I needed it.

Big thanks also go to Petr Brož for the proofreading he did for one of the attachments to this document.

I would also like to thank all the other people who have helped me during my studies and who have also kept me reasonably sane. Mostly, my deepest thanks go to p. Vladimír Slámečka who has always been there for me and Kateřina "Kačka Blátotlačka" Kuglerová – how she still has patience with me after I keep asking her about things I am supposed to know every single day is beyond me.

Obviously, I appreciate doc. Ing. Josef Vedral's willingness to become my project supervisor to the greatest extent.

There are many other really awesome people on ČVUT, and I simply cannot name all of them. This being said, I strongly believe that Prof. Ing. Pavel Sovka, CSc. and Prof. RNDr.Pavel Pták, DrSc. both deserve at least my honorable mention.

I'd also like to thank Daniel "Kokon" Tichý for helping me with learning Java back in the days (and also gifting me that plushie of Pinkie Pie pony, although I have no idea what made him do that).

Finally, I would also like to thank Oxana Kovbasjuková for giving me that amazing rooster plushie for Christmas, since it's really awesome. Thanks a lot!