M.E.D.K.I.T

Medical and Embedded Device Konfiguration and Integration Technology

---

Conál (*angl.* Connell) **Paxton** | **Evangelos Loukedes** | **Jaime Santoro**

https://github.com/connellpaxton/medkit/

The Team

Conál: Embeddded & Network Software, Systems-level Design, Documentation

Jaime: Power Supply Hardware Design, Hardware Procurement

Vagos: Casing Design

Problem: Healthcare is Expensive[1]

[1]for Americans.

Medical Debt

- pervasive and catastrophic
- Negatively influences availability of treatment options to patients.
- Most Americans have three choices:
  1. Be born into enough wealth to never worry about it
  2. Have employer-based coverage
  3. Take on massive amounts of risky debt to cover treatments
- What happens when you run out of money (and your car, house, etc)?

Amount	# of people
Some	20 million
>$1,000	14 million
>$10,000	3 million

Case Study: Insulin Pumps

- Up-front ~$6000 cost (plus additional subscriptions, etc)
- Cost manipulation of Insulin
- Only Real Alternative: **Cost Related Nonadherence**
  - paradoxically, more expensive
  - often kills you[2]

[2] After taking your vision, mobility, and your limbs once they get amputed.

Solution: Open Source BioTech?

Solution: Open Source BioTech

Advantages

- Much cheaper
- Shares information, paves the way for future work
- Decentralized development allows diverse forks and features

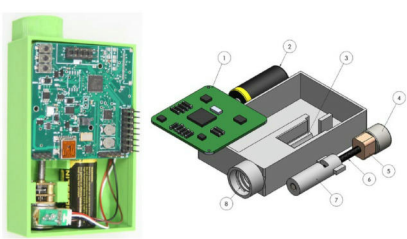
Disadvantages

- Harder to get standardization without the backing of Capital
- Capital is disincentivized to engage or support projects due it helping people that may not directly give you money.

Ultra-Low-Cost Insulin Pump

Pros

- $87 is a lot lower than $6000
- High-Quality
  - Compettitive (and sometimes superier) bench-side delivery accuracy.



Cons

- It doesn't work with my $700 Glucose Monitor!

Integration

Integration

Why?

- Control Systems!
  - having diabetes is a full-time (reverse-finnanced) job!

How?

- Communication between medical devices according to a shared communication policy.

But...

- Existing or non-compliant devices are inacessible
- Future designs would like to have things to connect to
  - MAJOR barrier to adoption

Solution: M.E.D.K.I.T!

(I) The Protocol

Set of rules to allow devices to connect to each other.

Should be **flexible** enough to incorporate arbitrary medical device systems of reasonable **configuration**, and accomadate **diverse feature-sets** through modular design.

(II) The Bridge Device

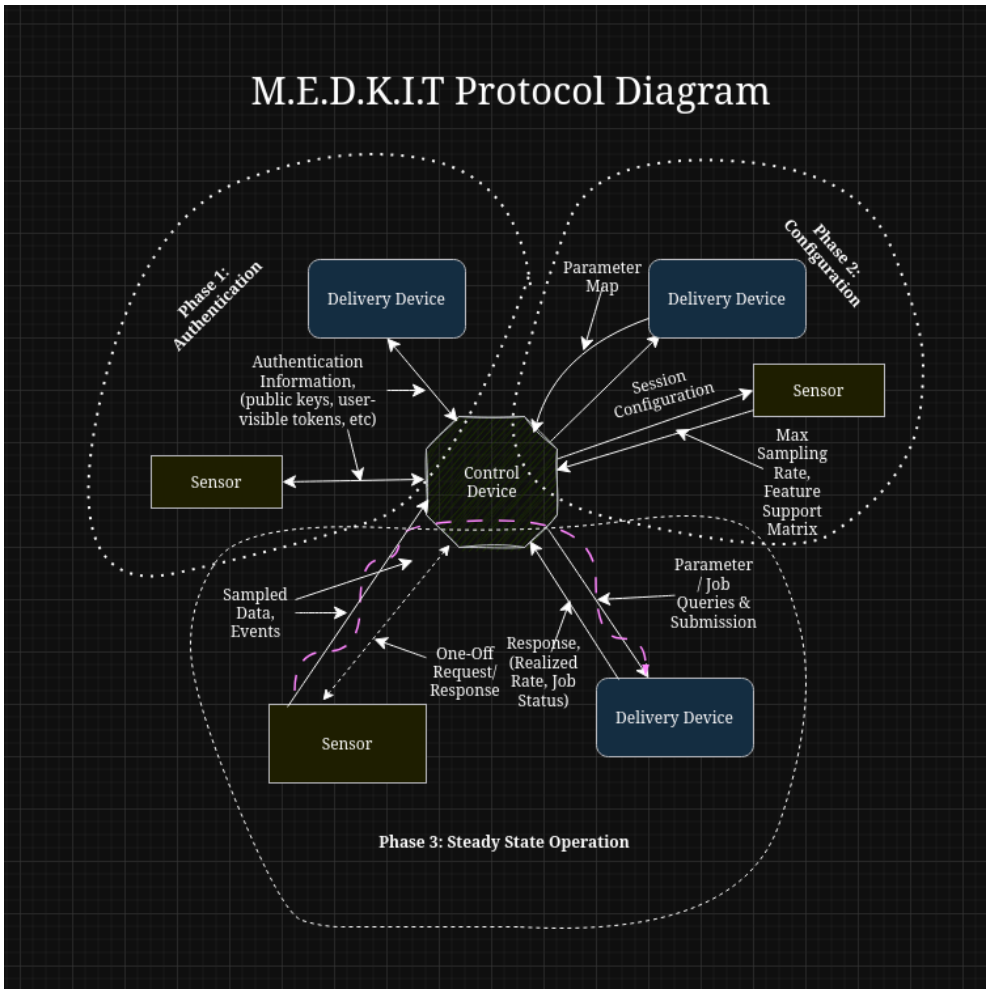
A **physical** device that allows a large subset of devices that do not implement **The Protocol**.

Exploits existing (mechanical or electric) user interface of the device. Allows for **1st or 3rd parties** to extend a device without redesigning hardware.

(I) The Protocol

Protocol: Overview

- 3 “Roles”
- 3 Stages
  1. Authentication
  2. Configuration
  3. Steady State
- Flexibility through “Feature Sets”



Implementation

- Reference Implementation: C++
  - No STL or stdlib dependencies - key for embedded
  - NetworkInterfaceTrait
- Lots of abstraction, but low memory footprint
- Configuration Table Layout: access-time-optimized
- High-Frequency Packet Layout: size-optimized

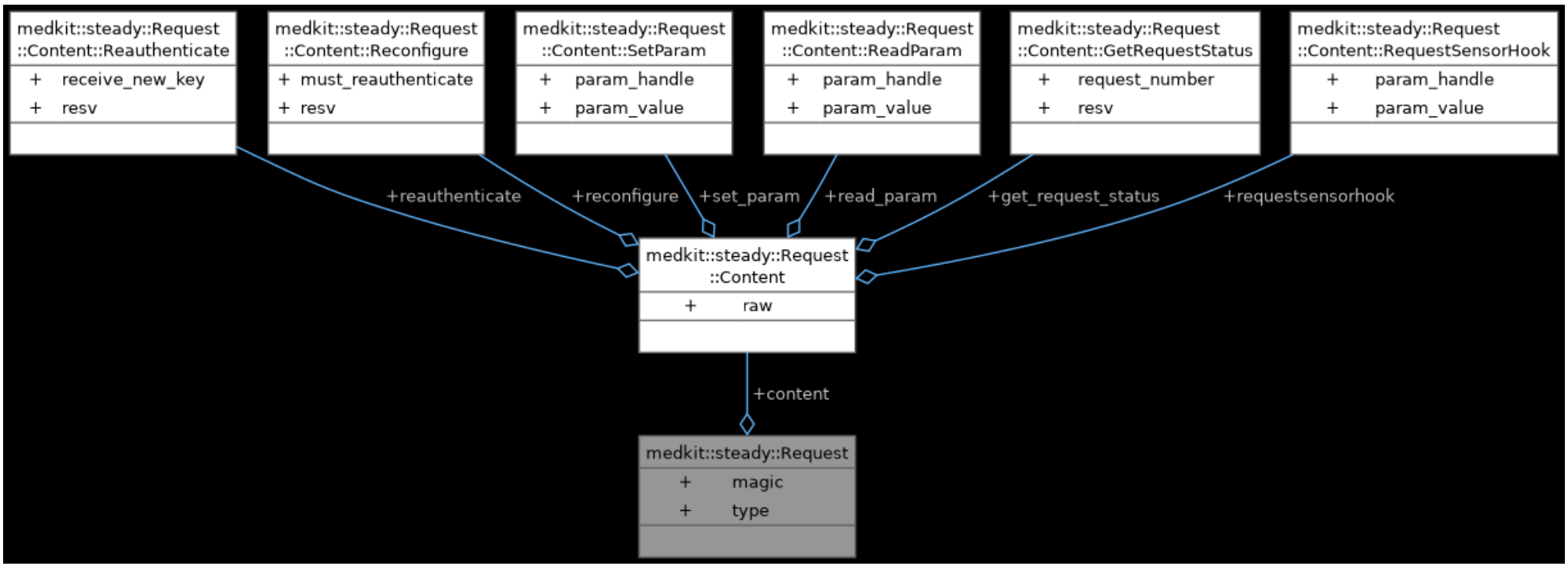
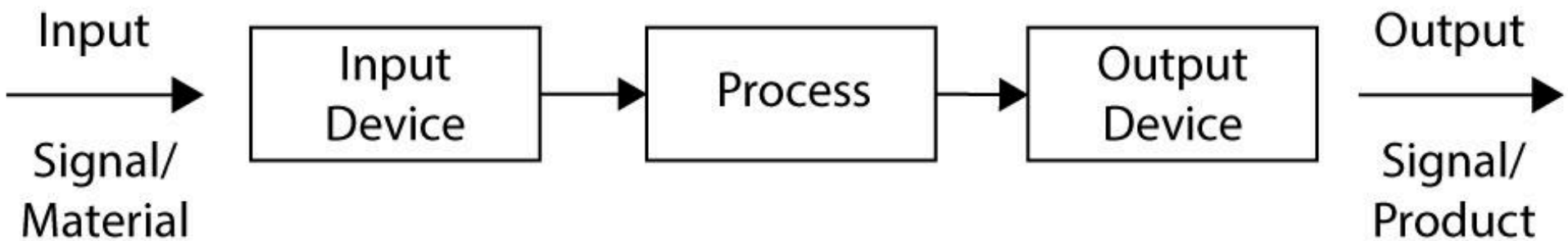


Figure 1: C++ Struct Diagram for the Request Packet

(II) The Bridge



Intuition: The Problem

- Systems are just arrows and boxes!
- The protocol governs boxes with some types of arrows

The Bridge: The Problem

- Not every system has the arrows you want
- You can't really just add arrows to a box
  - faaaaaaar too system-specific for a single protocol to manage
  - requires extremely in-depth information about internal arrows

Intuition: The Solution

- Put a box around the problem
- Draw your own arrows
- Use whatever you know about the system to draw internal arrows
  - When you know nothing, think like a user

Bottom Line: We can own the arrows ***and*** the boxes.

Implementation

- Everything has a button…

Implementation

- Everything has a button
- Every button has a wire…

Implementation

- Everything has a button
- Every button has a wire
- Wires don't really know that they're connected to buttons…

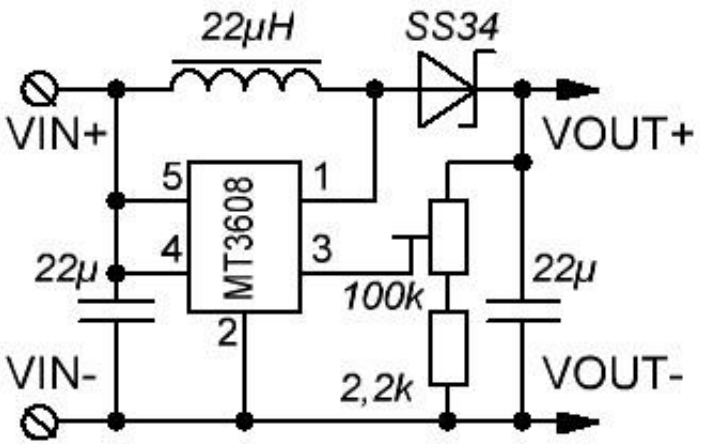
Implementation

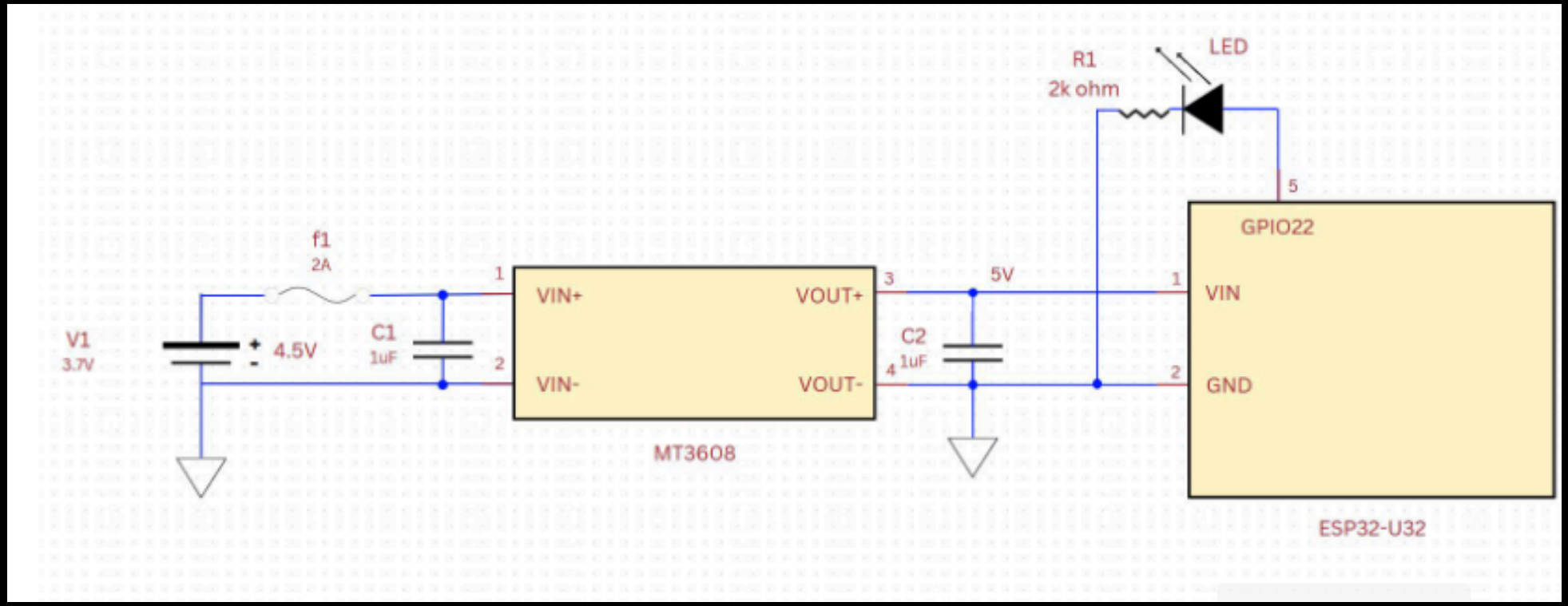
- Everything has a button
- Every button has a wire
- Wires don't really know that they're connected to buttons
- (Delivery) We can splice in wires into input streams and fake our own inputs
- (Sensor) We can splice in wires into output streams and translate dispatch information.
- Due to the flexible nature of the protocol, we don't have to do much to reach compliance.

Hardware

Hardware: Power Supply

- 3.7V, 1A Lithim Ion Battery
  - previous design: 2 AAA batteries
- MT3608 Power Module (Boost Converter)

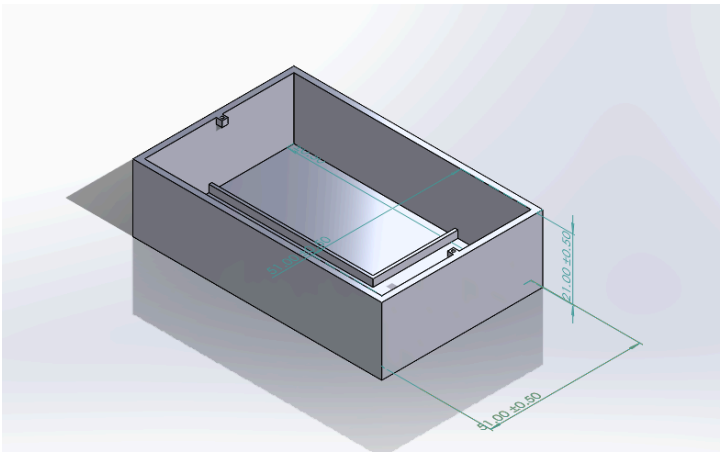


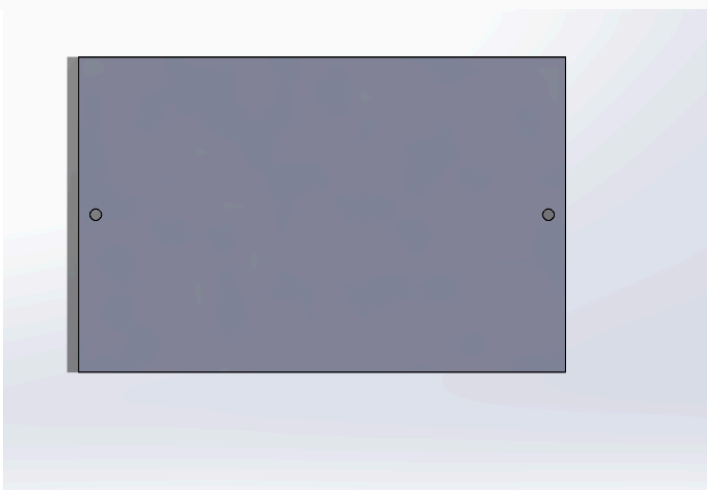

R1
2k ohm
LED
5
GPIO22
f1
2A
1
VIN+
VOUT+
3
5V
1
VIN
V1
3.7V
4.5V
C1
1uF
2
VIN-
VOUT-
4
C2
1uF
2
GND
MT3608
ESP32-U32

The Bridge: Casing

The Bridge: Casing

- PLA Plastic
- Seperate Dividers
- Open-Top Design
  - Easy Replacement
- Dense Filiment





Demo

Demo

- Three devices in typical sensor-control-delivery topology
- Configuration:
  - Sensor (ESP8266 with button), with message type NOTIFY
    - real system analog: heart monitor with alert set for threshold
  - Deliver (ESP8266 with LED), set with SET_PARAMS, read with GET_PARAMS

https://www.youtube.com/shorts/M94XlU5okJ0

ROI

- Standardizing Body