The History of Loop and LoopKit

Reflecting on the past in celebration of version 1.0.

Nate Racklyeft · Follow

9 min read · Oct 2, 2016

Part 1: Vision

On July 21, 2015, I first placed my trust in a collection of Python scripts I wrote to automate my insulin needs for Type 1 diabetes. Every 5 minutes since then—while I've worked, slept, vacationed, commuted—code has been the primary interface to my treatment of this chronic condition.

The first system to run it was a Raspberry Pi; the code was a series of plugins, written with the help of Chris Hannemann, to the openaps toolkit developed by Ben West in collaboration with Dana Lewis and Scott Leibrand. I'm still in awe of the elegant premise in Ben's design: a system of repeatable, recordable, and extendable transform commands, all backed by Git. The central plugin of the toolkit is decocare: Ben's 5-year *magnum opus,* a reverse-engineered protocol of the Minimed Carelink USB radio to command insulin pumps.

Pete Schwamb had a vision for an entirely different system, and the skill-set to bring it to life. He designed a clone of the Carelink radio and bridged it to a Bluetooth Low-Energy interface, opening the door for a diverse set of BLE-compatible devices, like his iPhone, to talk to an insulin pump.

Pete Schwamb on Twitter

@danamlewis Maybe someday it will look like this! :)

Master of many things, including time travel.

• • •

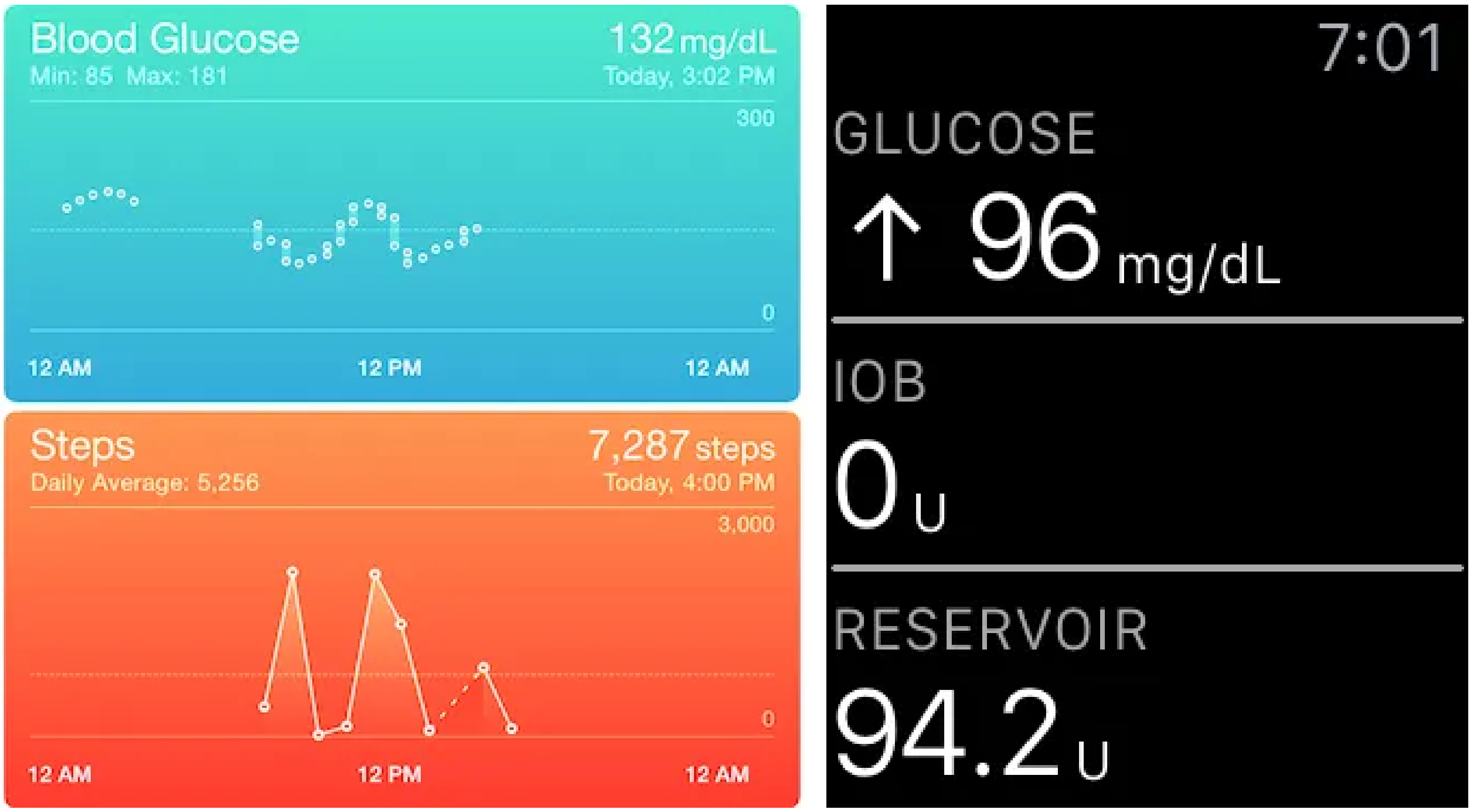
I quickly co-opted Pete's vision as my own. I got my first "RileyLink" the same week I started "looping" with the Raspberry Pi. "I'm done," I told my wife, "now I'm going to rebuild it all."

Pete had written a significant portion of the initial Bluetooth code in an iOS app which captured and uploaded Minimed sensor values to Nightscout, but there were still a lot of infrastructure questions to validate:

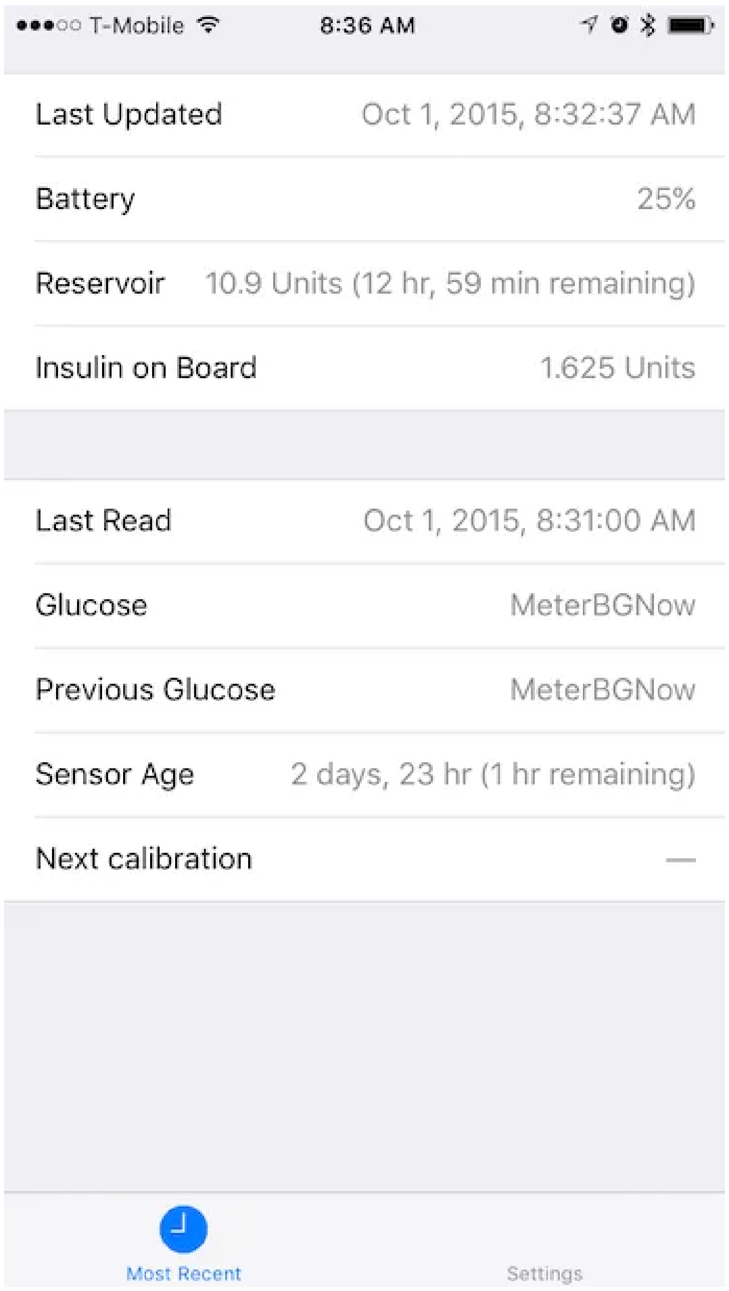
1. Will a background app run reliably on iOS?
2. What will the battery impact be?
3. How much data should be stored, and where?

Any large project is going to face questions like this. One thing that's served me well in the past is to avoid answering the foundational framework questions as long as possible. Often these questions answer themselves as the upper parts of the stack take shape. Months later, a colleague would share this great post on Lug-Nut Development that described this philosophy so perfectly.

So, using Pete's RileyLink code, I took my sensor values and started putting them into UI, using Apple Health and a simple Apple Watch app.



I deferred most app architecture while building HealthKit and watchOS 2 integrations



Last Updated	Oct 1, 2015, 8:32:37 AM
Battery	25%
Reservoir	10.9 Units (12 hr, 59 min remaining)
Insulin on Board	1.625 Units
Last Read	Oct 1, 2015, 8:31:00 AM
Glucose	MeterBGNow
Previous Glucose	MeterBGNow
Sensor Age	2 days, 23 hr (1 hr remaining)
Next calibration	—

The app displaying transient data from the RileyLink

Though my intention was to try to make the small amount of data I had useful, I also inadvertently learned the quirks of the HealthKit and WatchConnectivity frameworks which would prove informative much later in architecture design.

Two much larger roadblocks appeared first:

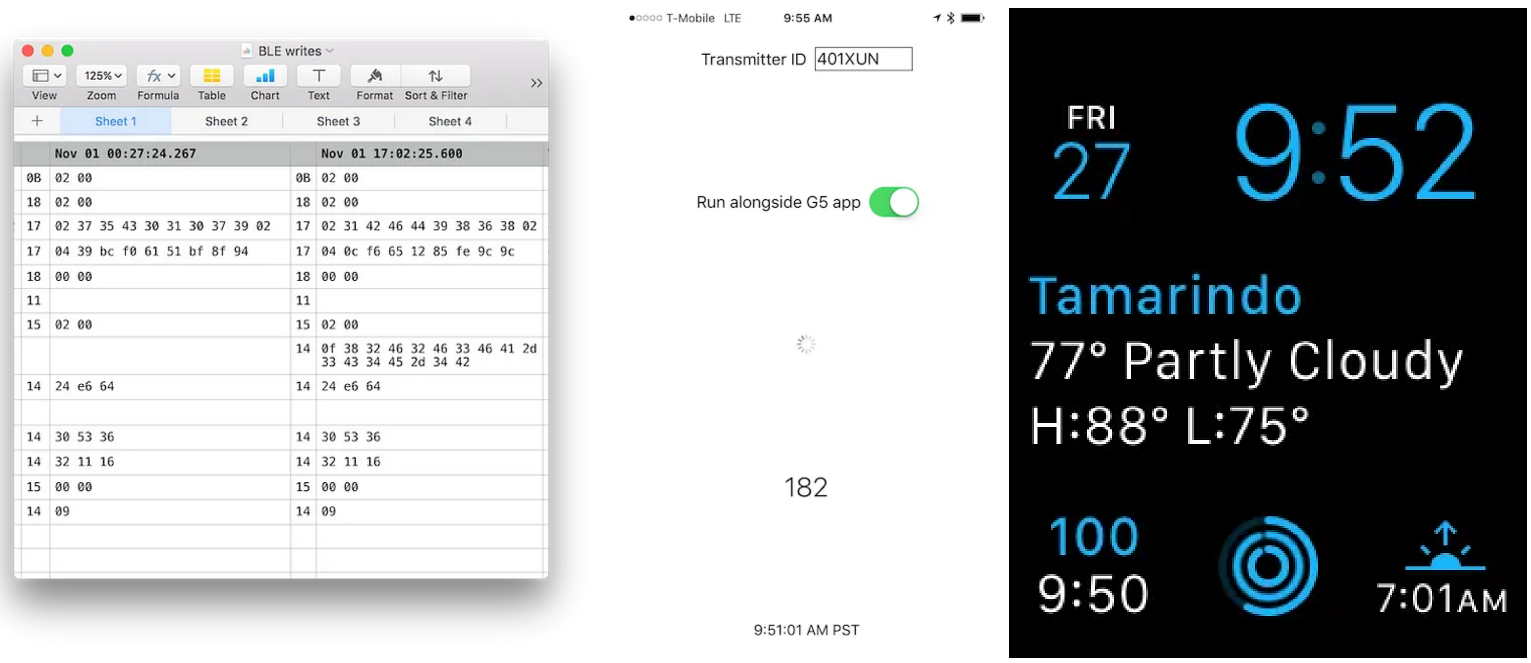
1. The original RileyLink crystal oscillator couldn't create the right frequency for longer radio messages needed to control a pump.
2. I received one of the first shipments of the Dexcom G5 CGM and nobody had any idea how to connect to it.

. . .

Pete discovered and diagnosed the frequency issue, and like a champion, offered folks to ship him back all of the dozens of existing boards. He replaced each crystal on his reflow workstation, and tested each board by sending a "button press" command to turn on the screen backlight, the delight of each flicker rivaling Edison's bulb. He can tell you the rest of that story.

. . .

The G5 CGM was a critical piece of Pete's vision to minimize the carry bulk. But a new device meant a new protocol to decode, and a very small user base available to collaborate. In addition, I had a hard release date I needed to hit: a trip to Costa Rica. Apple Watch could certainly handle the tropical humidity better than my iPhone, and the goal was to read G5 glucose data directly from the transmitter and shuttle it onto the watch, without the use of an internet connection, in time for the vacation.



	Nov 01 00:27:24.267		Nov 01 17:02:25.600
0B	02 00	0B	02 00
18	02 00	18	02 00
17	02 37 35 43 30 31 30 37 39 02	17	02 31 42 46 44 39 38 36 38 02
17	04 39 bc f0 61 51 bf 8f 94	17	04 0c f6 65 12 85 fe 9c 9c
18	00 00	18	00 00
11		11	
15	02 00	15	02 00
		14	0f 38 32 46 32 46 33 46 41 2d 33 43 34 45 2d 34 42
14	24 e6 64	14	24 e6 64
14	30 53 36	14	30 53 36
14	32 11 16	14	32 11 16
15	00 00	15	00 00
14	09	14	09

Transmitter ID 401XUN

Run alongside G5 app

182

9:51:01 AM PST

80’s musical montage of November 2015: from sniffing, to prototyping, to the beach

Goal met.

Nate Racklyeft on Twitter

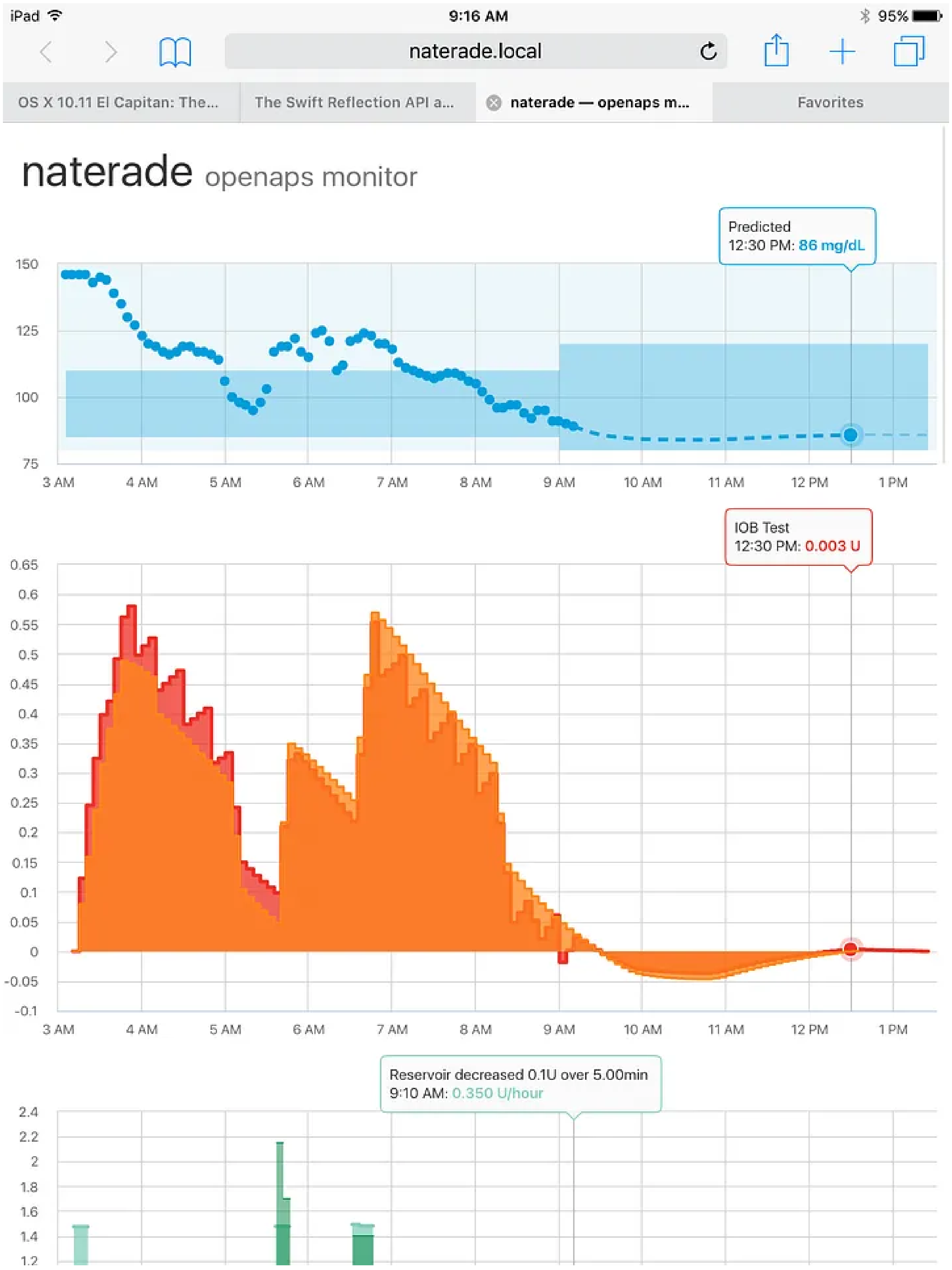
Happy new year! xDripG5 is now available using Carthage and @CocoaPods. Build cool apps with live glucose data! https://t.co/hkdtrxj97S

. . .

With an upgraded RileyLink and G5 data flowing, it was time to start to consider those architecture questions, starting with battery and background time.

There’s two generations of insulin pumps which can accept remote commands. The later generation has a feature which broadcasts its status every 5 minutes, which is the basis of the MySentry and Connect products. (The latter fulfills essentially the same function as the RileyLink, though it was released to the public later). Capturing this broadcast would mean fewer power-hungry communication sessions with the pump, reserving resources only for the times when a dosing change was necessary.

This broadcast includes the pump’s reservoir volume, which, when stored over time, should be sufficient to know how much active insulin was currently in the body. To validate this premise, I went back to my Raspberry Pi and started capturing and experimenting with this data source, simulating the lower resolution (0.1 U) values compared to what was available in pump history (0.025 U).



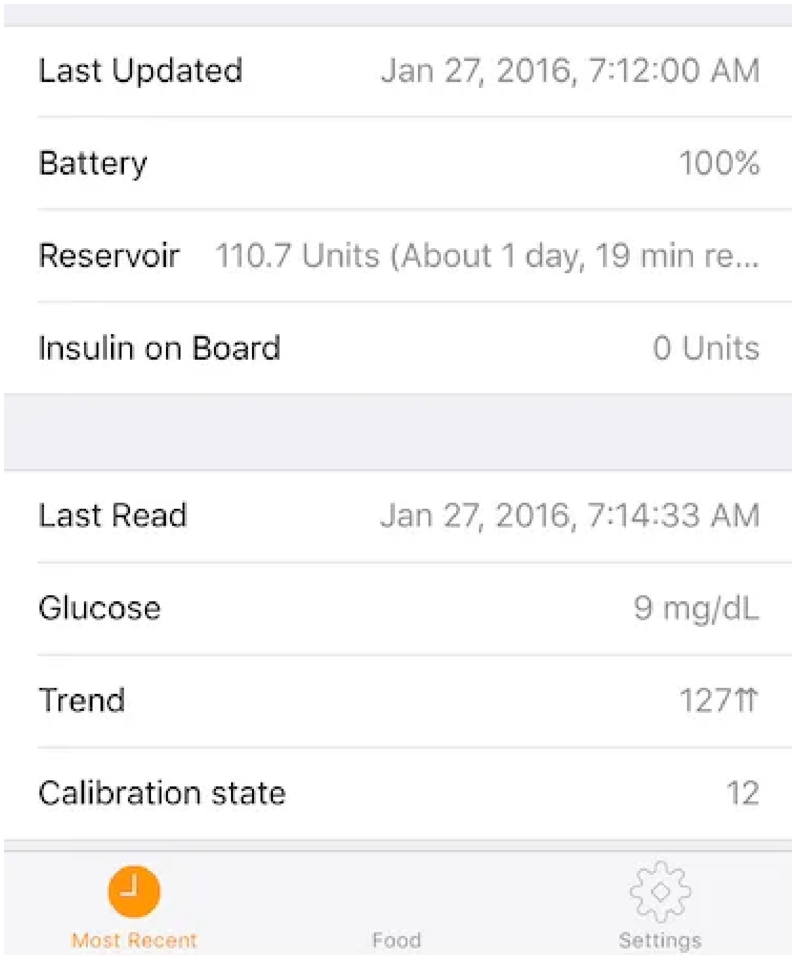
An early test comparing IOB from reservoir to IOB from pump history. The sawtooth pattern is an effect of the simulated resolution difference.

This visualization was extremely useful, and exposed a bug or two my unit tests missed.

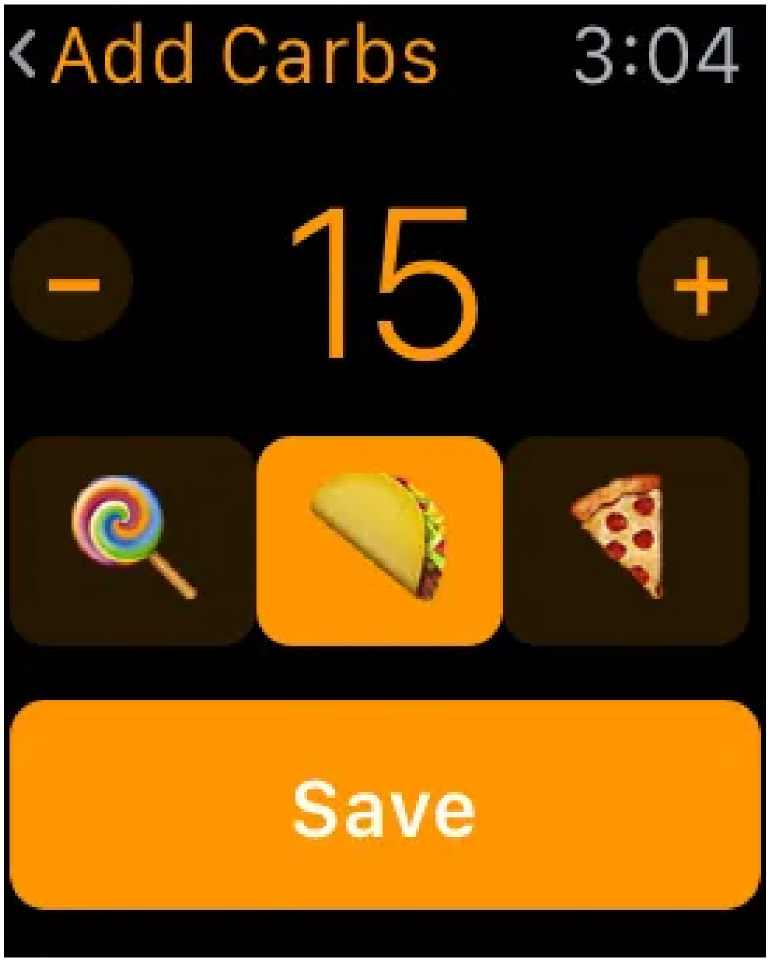
. . .

With blood glucose and insulin data ready, the final input to the equation was carbohydrates. I wanted to use HealthKit for this data, as there are so many great meal and ingredient databases in the App Store that store carbohydrates in HealthKit. It seemed like a perfect match to be able to use them right in my app's algorithm.

At the same time, I couldn't imagine a better input mechanism for entering food than my watch. Building both ends simultaneously surfaced some unexpected challenges around encryption states in HealthKit.



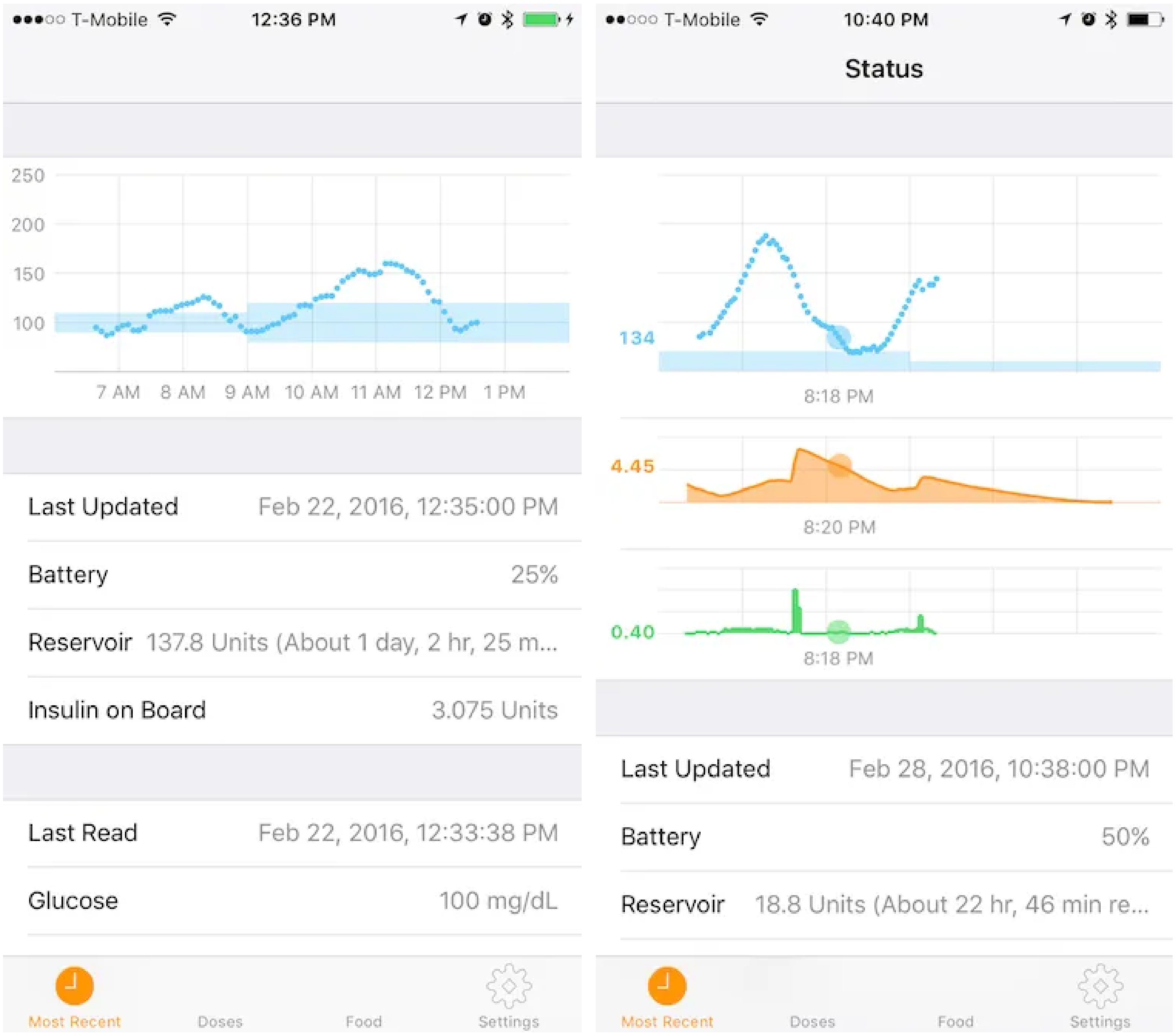
Last Updated	Jan 27, 2016, 7:12:00 AM
Battery	100%
Reservoir	110.7 Units (About 1 day, 19 min re...
Insulin on Board	0 Units
Last Read	Jan 27, 2016, 7:14:33 AM
Glucose	9 mg/dL
Trend	127⇈
Calibration state	12



Food was the final input. Note that the CGM was in a bad calibration state; glucose wasn't actually 9 mg/dL.

By early February, I could input all the data I needed, and I'd started exploring storage and retrieval. I was also getting a better picture of the phone's battery life: ~4 hours of daily background time, consuming ~15% of a full charge.

Validating the algorithms I originally wrote in Python started with porting over the suite of unit tests (using the same JSON fixtures in both codebases), but it also required me personally trusting all the points of integration that were harder to test. I knew I could achieve that trust by simultaneously building the graphs as I built the algorithm.



Status

Last Updated	Feb 22, 2016, 12:35:00 PM
Battery	25%
Reservoir	137.8 Units (About 1 day, 2 hr, 25 m...
Insulin on Board	3.075 Units
Last Read	Feb 22, 2016, 12:33:38 PM
Glucose	100 mg/dL

Last Updated	Feb 28, 2016, 10:38:00 PM
Battery	50%
Reservoir	18.8 Units (About 22 hr, 46 min re...

The graphs took shape simultaneously with the data storage and analysis

For the next two weeks, the iOS loop was running alongside the Raspberry Pi, and I was continuously comparing the visualizations and looking for edge cases. On March 13, I shut down the Raspberry Pi (to never turn it back on) and gave control to my app. I teased the moment on Twitter:

Nate Racklyeft on Twitter

Evolution #WeAreNotWaiting https://t.co/N2ylomLqsN

Symbolizing the 15 years since my first insulin pump, on the eve of the next phase.

Once I was sure it was working, I started showing off.

Nate Racklyeft on Twitter

I'm now at 24 straight hours of iOS looping powered by @ps2's RileyLink, xDripG5, and LoopKit!

Nate Racklyeft on Twitter

Breakfast bolus from my Apple Watch https://t.co/ur62ic8yxB

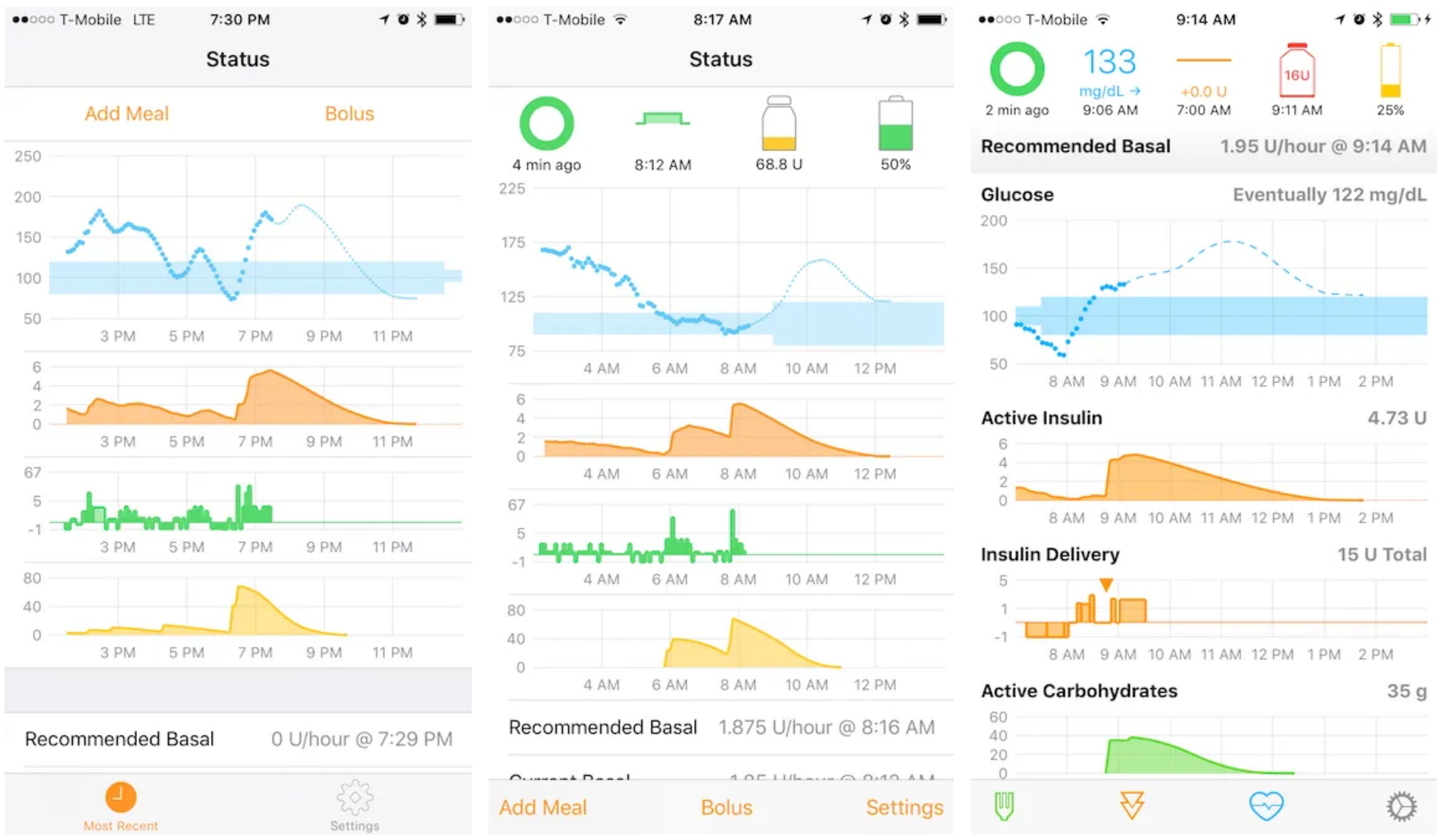
Part 2: Sharing

As the celebration faded, many wise people asked me if I was done. Those who watched me trade time with family, friends, and every hobby I had to pursue this project were right to challenge me on what I'd do next. I wanted to listen, but I also didn't know how to stop.

. . .

It was clear that just a collection of frameworks—LoopKit, MinimedKit, xDripG5, and RileyLinkKit—weren't enough to help others use the project. While the containing app was also open source, I cautioned that it should be used only as an example. This advice was understandably ignored by others: iOS development has a barrier of entry larger than most can tackle in their free time.

What that meant was that the app, now known as simply "Loop", would need to expand in device support, safety features, and remote connectivity to satisfy the needs of prospective users. Some of this was graciously taken on by Mark Wilson (G4 glucose via Bluetooth & HTTP) and Pete Schwamb (Nightscout monitoring). Instead of slowing down, my time spent committing code and supporting users continued at the same level it was during initial development.



The evolution of Loop's status screen

There were many proud moments.

Ali Mazaheri on Twitter

First day at school with Loop @loudnate @ps2 @bewestisdoing

Howard Look on Twitter

Ready for the first day of school with Loop and RileyLink. Thanks @loudnate @ps2 @OpenAPS

Eric Matthews on Twitter

Loop with new #rileylink helping my daughter manager her #T1D during school. Thanks @ps2 and @loudnate

Katie DiSimone on Twitter

Today we started #Loop. Tested it last night with a not-low-carb meal and wow! I could cry happy tears @loudnate @ps2 THANK YOU

Part 3: Burden

In my setup instructions for Loop, I told users,

> *You might open this app a lot. Make it the most personal app on your iPhone by changing the name and icon.*

It *was* a personal app for a lot of people, and the questions, opinions, and imperatives rolled-in from users and non-users alike in the chat room. Thanks to users in far-off time zones, there were always new messages to read. It didn't take long before I felt overwhelmed. Thankfully long-time users tried their best to answer as many questions and criticisms as they could, allowing me to focus the time I had on coding fixes.

. . .

It hadn't realized that passion projects could outlive their passion, but it happens every day in open-source projects. Loop was going to be an endless "loop" of development unless I took steps to slow down.

I thought of Loop as "My" project. It was my rejection of Design-By-Committee open-source. At times, it didn't play well with others' ideas (even when they were good). I never pre-announced features, or created spaces for feedback. I shouldered all of Loop's secrets, as well as all of its criticisms. Neither were a burden anybody asked me to bear. I needed to relinquish control of Loop to regain control over my time.

Part 4: 1.0

Version 1.0 is an end-cap to a difficult but rewarding era of development. The project—its features, faults, and future—now belongs to the community of users. Pete Schwamb and Mark Wilson have graciously agreed to coordinate pull requests and code reviews. Chris Hannemann will lend a hand to coordinate features and issues. They are ready and eager to nominate additional collaborators.

The promise of an automated insulin delivery system is to reduce the time and cognitive load spent on diabetes, and Loop seeks to deliver it with its glanceable interface, seamless connectivity, and the compact package of the RileyLink.

It's time for me to finally realize that promise personally: I'll be stepping down from development after the release. My passion for health and wellness projects will continue, professionally, as I join the team at Apple.

. . .

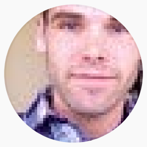
I want to thank the developers and tinkerers who've been tackling Type 1 diabetes with software and hardware for years. You can ask Scott Hanselman who they are: they paved the way for Ben West to make the single-greatest patient innovation in Type 1 diabetes. Thank you to Pete Schwamb, whose unmatched ingenuity led to solutions for the most difficult problems. Thank you to Chris Hannemann, Ken Stack, and Gustavo M. whose algorithmic and instructional expertise was foundational to the most critical code in my projects. Thank you to Dana Lewis and Scott Leibrand for creating a vibrant community around artificial pancreas devices.

Most of all, thank you to the Loop users for your passion and trust. I can't wait to see what you build next.

Loop and LoopKit: Version 1.0

medium.com

Diabetes Healthcare



Written by Nate Racklyeft

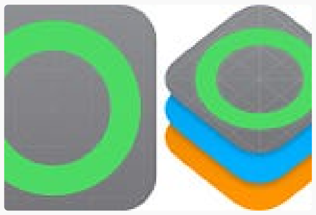
127 Followers

Follow

More from Nate Racklyeft

Nate Racklyeft

Loop and LoopKit: Version 1.0



Loop and LoopKit have reached 1.0 after just over a year in development. Loop uses RileyLink, a CGM, and an insulin pump to automate...

1 min read · Oct 2, 2016

21 1

See all from Nate Racklyeft

Recommended from Medium