#OpenAPS

# OpenAPS Documentation

***Release 0.0.0***

**Ben West, Dana Lewis, Scott Leibrand, openaps community**

June 06, 2016

Contents

CHAPTER 1

# Introduction

Introduction section includes:

- Understanding This Guide
- Ways to Contribute
- Where to Go For Help With Your Implementation

## 1.1 Understanding this guide

Some conventions used in this guide:

- Wherever you see text that is formatted `like this`, it is a code snippet. You should copy and paste instead of attempting to type this out; this will save you debugging time for finding your typos.
- You will see a $ at the beginning of many of the lines of code. This indicates that it is to be entered and executed at the terminal prompt. Do not type in the dollar sign $.
- Wherever there are `<bracketed_components>` in the the code, these are meant for you to insert your own information. Most of the time, it doesn't matter what you choose **as long as you stay consistent throughout this guide**. That means if you choose `Barney` as your `<my_pump_name>`, you must use `Barney` every time you see `<my_pump_name>`. Choose carefully. Do not include the `< >` brackets in your name.

### 1.1.1 Before you get started

Some familiarity with using the terminal will go a long way, so if you aren't comfortable with what `cd` and `ls` do, take a look at some of the Linux Shell / Terminal commands on the Troubleshooting page and the reference links on the Technical Resources page.

One helpful thing to do before starting any software work is to log your terminal session. This will allow you to go back and see what you did at a later date. This will also be immensely helpful if you request help from other OpenAPS contributors as you will be able to provide an entire history of the commands you used. To enable this, just run `$ script <filename>` at the beginning of your session. It will inform you that `Script started, file is <filename>`. When you are done, simply `$ exit` and it will announce `Script done, file is <filename>`. At that point, you can review the file as necessary.

### 1.1.2 What you won't see in this guide

You won't see a full loop where you can just download the code, press a button, and have a live loop. There are many places where there are examples, and instructions, but you must do the work to understand how to communicate

between devices and transfer data between reports and files. This is key for helping you understand what you are building and how it will work.

In some cases, the documentation needs to be built out further, with easier to understand language and more examples. However, there are a few things (like a full `cron` example) that are not included in this guide, and intentionally so in order to ensure that you have full intent and autonomy in building your system for yourself.

### 1.1.3 But wait - I need the "Dummy" version

Well, actually, you don't. If you can deal with diabetes, invoking shell scripts isn't all that big a deal. But it will probably help to have some idea what is going on here.

It may help to think of the OpenAPS setup as a tiny "diabetes brain" which is focused only on figuring out how much basal insulin you should be getting. It does this by collecting all that background data we usually let our pumps deal with: what is your insulin sensitivity, your target BG, the duration of action for insulin, etc. Then, it collects more immediate data, such as what is the current IOB, and basal rate, as well as checking out the CGM to see what your BG has been up to recently. Then, it decides what should be changed (if anything) and tells your pump to go to a new temp basal rate, either higher or lower, depending on all the other factors.

As you go through the steps to run the manual loop, the longer-term data is collected in the "settings" directory, with file names indicating what sort of data they contain. The data representing what is going on right now is in the "monitor" directory, and the recommendation for what should change goes in the "enact" directory. To close the loop, you add a "cron" script, which just directs the computer to do something at a certain time interval. You will need to be very certain all your manual pieces are running correctly before you decide to close the loop, which is why the "cron" section does not contain any examples.

## 1.2 Ways to Contribute

OpenAPS doesn't require you to be a formally trained engineer/developer/anything to get started or use these tools. The main requirement is interest and willingness to safely DIY new technology that may help improve your life as well as others.

If you're not sure where to get started, here are some ways to get involved:

- Do a fresh install using this guide and see where you get stuck; if you have to do something off-script, there is a reasonable chance the script is either wrong or your case is "special" and should be accounted for here. Make edits and submit a pull request to change the document and assist others.
- Additionally, this guide needs more work, always. If there is something that is not documented, do one of two things: a) submit a pull request (see above). or b) log an "issue" (go here to see the open issues) about the section or thing that needs more documentation.
- Ask questions on gitter; if your question wasn't answered in this doc and gets answered there, chances are it should be included here—go ahead and add it to the appropriate section.
- Test the openaps tools for different use cases and report back with your findings. Log files and, if you're comfortable, associated device data (CareLink CSV files, for example) are extremely helpful for debugging.
- Submit issues on GitHub and work with other contributors to get them resolved.
- Develop a plugin to enhance the functionality or ease-of-use of openaps.
- Spread the word about #OpenAPS and get others involved; the more, the merrier. (You can direct them to OpenAPS.org for more information.)
- Consider calling your device manufacturer and ask about communication protocols in order to understand how your device operates.

If you would like to work on the core openaps code, take a look at the openaps contributing guidelines before getting started.

## 1.3 Where to go for help with your implementation

There are several ways to communicate with other participants and contributors in the #OpenAPS project. See also the Resources section for additional assistance.

**Note:** It's best practice not to share your pump's serial number, so make sure not to include it in pictures or pasted text output when seeking help on pump communication.

### 1.3.1 Gitter

Gitter is a messaging/chat service similar to IRC. It provides integration with GitHub and several other services.

- The nightscout/intend-to-bolus channel is where you will find active #OpenAPS discussions ranging from technical issues with openaps tools to control theory to general information. It is a great place to introduce yourself and get some help from those who are a few steps further down the road.
- For TI stick communication, use the oskarpearson/mmeowlink channel
- For RileyLink conversations, use the ps2/rileylink channel
- For LoopKit conversations, use the loudnate/LoopKit channel

### 1.3.2 Google Groups

A private google group focused on #OpenAPS development work can be found here. Request access to participate and see some of the archived discussions. If you're new, make sure to introduce yourself!

### 1.3.3 Issues on openaps GitHub

For reporting issues on the openaps tools formally, the openaps issues page on GitHub is the proper forum. Feel free to try and get through the issues by working with others on the Gitter channel first if you think it may be something unrelated to the codebase.

### 1.3.4 Other online forums

Those in the #OpenAPS community are frequently found in other forums, such as on Twitter (using the #OpenAPS hashtag, as well as #WeAreNotWaiting) and on Facebook in the "CGM In The Cloud" group.

There is also a Slack channel to discuss communication around other pumps that are being explored for being used for other DIY closed loops.

CHAPTER 2

# Guide to understanding openaps

Welcome to openaps

## 2.1 Installing openaps

The recommended way to install openaps is to use python's package management system. The openaps project is distributed on pypi. The following apt-get dependencies are required (they can be installed through variety of means, in debian/ubuntu and apt based systems the following packages are required:

```
sudo apt-get install python python-dev python-pip python-software-properties python-numpy
sudo pip install setuptools
```

### 2.1.1 From pypi

To install from pypi:

```
sudo easy_install -Z openaps
```

This installs `openaps` system wide.

#### Updating

To update `openaps`, append the `-U` option:

```
sudo easy_install -ZU openaps
```

### 2.1.2 From source

Sometimes, it's useful to use a development version to help test or debug features. Here's how to install the project from source.

```
git clone git://github.com/openaps/openaps.git
cd openaps
# checkout desired branch (dev)?
git checkout dev
sudo python setup.py develop
```

Do not use `openaps` commands in the the openaps repo. Only use the `openaps` directory for hacking on the core library, or for managing upgrades through git. Running `openaps` inside of the openaps source directory will error in the best case, and mess up your `openaps` install in the worst case.

## 2.2 About openaps

`openaps` is a toolkit to make developing an artificial pancreas easy for humans. What is an artificial pancreas? What are the requirements of this toolkit and how does it work?

While most people learning about this project may be interested in assembling their own "AP" as soon as possible, it may be helpful to pause and get a high level overview of openaps.

### 2.2.1 What is an artificial pancreas?

The term artificial pancreas can be misleading, as it can include or exclude different ideas or projects. For our purposes, an artificial pancreas does the following things:

1. **Monitor** therapeutic data (especially from pumps and CGM) in real time to provide relevant decision making data points. In order to do this, we must be able to manage data from disparate systems in a **uniform** manner.
2. **Predict** what is likely to happen. For example, when administering insulin, we naturally expect glucose to fall along a given curve. High fidelity therapy means communicating expectations like this, learning when it is wrong, and responding accordingly. Since clinical therapy is an application of science, this process must be **reproducible**, so that we can properly assess what works, and what does not.
3. **Enact** shared plans like a good teammate. When humans get distracted, or sick, or are asleep, the tools can and should provide assistance to improve **safety** and increase **control** over dangerous conditions.

### 2.2.2 Bring your own device

Circa 2016-03-27, we have access to a variety of insulin pumps and glucose monitors. openaps provides a modular framework to manipulate a variety of insulin pumps, and glucose monitors, as well as **devices** by different **vendors**. openaps exposes the **uses** of a **device** from a particular **vendor** for you to explore your own menagerie of devices in a **uniform** and interoperable manner. `openaps` also provides `reports` to track data as it flows through each phase, and an **alias** feature to logically group commands and workflows.

When the community discovers how to communicate with a device, we can create an openaps **vendor** module for the new device.

### 2.2.3 What is a toolkit?

`openaps` is not an artificial pancreas by itself. So far in discussing some of the philosophy and overview of openaps, we've seen that it needs to interact with a lot of different devices in a lot of different circumstances. The solution to solving this problem was to provide a suite of tools that allows us to interact with the properties of the system, easily see what is going on inside and inspect the data. Because we often intend for the system to operate while we're asleep, the tools also provide several facilities for easily changing and saving configuration.

This means we'll be using the command line a lot, because it's very easy to teach, reproduce, and automate. Any series of commands you type in the command line can be put in a file and made into a script.

Because most people intend to use this as a medical device, we also wanted to track the data as it moves through the system. openaps checkpoints all the data so that it can be inspected and verified at anytime.

The toolkit provides many tools to realize the values enumerated by our design philosophy. Here are a few, but don't panic, we'll go through how to use each one methodically:

```
openaps
openaps-device
openaps-report
openaps-vendor
openaps-use
openaps-alias
```

Now that `openaps` is installed, we can ask any of these tools for help to explain themselves:

```
`openaps --help`
```

Many of the tools depend on some configuration though, how do we provide the configuration it needs? If it needs configuration, the tool might refuse to run, like this:

```
bewest@bewest-MacBookPro:~/Documents$ openaps use -h
Not an openaps environment, run: openaps init
```

Don't worry, it's time to create our first workspace!

### 2.2.4 My first workspace

This guide aims to help users become comfortable navigating the openaps commands. Most of the commands read saved configuration. `openaps` needs a location to store all the data and configuration generated as we explore. We'll often refer to this location as your **openaps instance**. The toolkit helps you create an **instance** of openaps on your computer. We can think about this location as a workspace for openaps.

Get familiar with `ls`, `cat`, `echo`, `cd`, `pwd` commands. To create our first openaps instance, we use `openaps init` with the location we'd like to use:

```
cd ~/Documents
openaps init tutorial-hello
cd tutorial-hello
pwd
```

This creates a new openaps **instance** in `~/Documents/tutorial-hello`. Take a look at what is in your workspace, using `ls` and `cat`. A valid instance must have an `openaps.ini` file. It can be empty, which simply means it doesn't (yet) know about your devices or vendors or anything.

---

**Note:** For the rest of this tutorial, all of our work will be done in the *~/Documents/tutorial-hello* directory.

---

### 2.2.5 Inspecting the log

We mentioned earlier that the requirements for `openaps` demand that we track data as it flows through the system. Is there a log of all the transactions that have occurred? If so, it should include the fact that we just created an instance of openaps. A valid instance of openaps has two requirements: it must be a `git` repository containing an `openaps.ini` file at it's root.

This means many operations are tracked using `git`, try `git log` or `git show`; there should an event in the log showing the time and date (and who!) created this instance.

```
bewest@bewest-MacBookPro:~/Documents/tutorial-hello$ ls
openaps.ini
bewest@bewest-MacBookPro:~/Documents/tutorial-hello$ cat openaps.ini
bewest@bewest-MacBookPro:~/Documents/tutorial-hello$ wc -l openaps.ini
0 openaps.ini
bewest@bewest-MacBookPro:~/Documents/tutorial-hello$ git show
```

```
commit 04715a67099c19ae220220d474aa67e470d07e0e
Author: Ben West <bewest@gmail.com>
Date:   Sun Mar 27 14:37:56 2016 -0700

    initializing openaps 0.1.0-dev

diff --git a/openaps.ini b/openaps.ini
new file mode 100644
index 0000000..e69de29
bewest@bewest-MacBookPro:~/Documents/tutorial-hello$
```

Knowledge of `git` is not usually needed or expected in order to use `openaps`, however, `openaps` does use it to store and track all data. This has several side-effects including easy mesh/backup. Many events in the log will also include a comment on the commands used to create the transaction. This allows us to share, find and debug problems quickly and easily.

Many software programs attempt to hide the inner workings. However because `openaps` has to meet exacting requirements, the design enables openaps to easily examine and adjust how it works using standard tools.

### 2.2.6 Summary

Congratulations, you're now the owner of a new openaps instance. It's time to start exploring `openaps` core in more detail.

#### Devices - aka using openaps

There are several **vendors** that built into the core of openaps. There are two tools we can use to examine which **vendors** are available for **use** by a **device**.

##### What can we use?

The `openaps use` command allows us to interact with devices. Let's ask it for help:

```
openaps use -h
```

```
usage: openaps-use [-h] [--format {text,json,base,stdout}] [--output OUTPUT]
                   [--version]
                   device ...
[... edited for brevity ...]
Known Devices Menu:
  These are the devices openaps knows about:

  device                Name and description:

Once a device is registered in openaps.ini, it can be used.
```

Notice the `Known Devices Menu:` is empty, this means `openaps` doesn't know about anything yet. Let's try the `openaps device` command instead:

```
usage: openaps-device [-h] {add,remove,show} ...

  openaps-device - Manage device configurations.

positional arguments:
  {add,remove,show}  Operation
    add              add - add a new device configuration
    remove           remove - remove a device configuration
    show             show - show all devices

optional arguments:
  -h, --help         show this help message and exit

show    - lists all known devices
add     - add a new device
remove  - remove a device
```

Lots of options there: the `devices` command allows us to teach openaps about our devices. The **use** command above had an empty devices menu what does `device show` say?

```
```

Nothing yet! Both `openaps use` and `openaps device show` indicate nothing for us to interact with yet. Let's look at the `--help` output for `device add`:

```
usage: openaps-device add [-h] [--extra EXTRA]
                          name {dexcom,medtronic,process,units} ...

add    - add a new device configuration

positional arguments:
  name

optional arguments:
  -h, --help            show this help message and exit
  --extra EXTRA, -e EXTRA
                        Name of extra ini file to use.

## VendorConfigurations:
  {dexcom,medtronic,process,units}
                        Operation
    dexcom              Dexcom - openaps driver for dexcom
    medtronic           Medtronic - openaps driver for Medtronic
    process             process - a fake vendor to run arbitrary commands
    units               Units - units tool for openaps
```

Notice `VendorConfigurations`. These are the default **vendors** that ship with `openaps`. However, `openaps` doesn't know we want to **use** them yet. `openaps device add` allows us to name our device. The name we *add* to openaps will be the name we **use** later.

#### Devices configure use

**A trivial device** Let's use `echo` to create a trivial device that just says "hello world." `echo` is a unix **process**, the **VendorConfigurations** above include a `process` vendor we can use to illustrate the relationship between **uses** and **devices**. Let's warm up with some examples:

```
echo 'hello world!'
```

That should print hello world on the screen. We can run this over and over again just for fun, but let's discover how to teach `openaps` how to do this.

```
openaps device add howdy process  echo 'hello world!'
added process://howdy/echo/hello world!
```

What did this do? Let's check `git show`:

```
commit 8e198ad8556ea6df4d4f6459d212eee316b89a0e
Author: Ben West <bewest@gmail.com>
Date:   Sun Mar 27 15:45:16 2016 -0700

    openaps-device add howdy process echo hello world!

          TODO: better change descriptions
          /usr/local/bin/openaps-device add howdy process echo hello world!

diff --git a/openaps.ini b/openaps.ini
index e69de29..d4a23d0 100644
--- a/openaps.ini
+++ b/openaps.ini
@@ -0,0 +1,4 @@
+[device "howdy"]
+vendor = openaps.vendors.process
+extra = howdy.ini
+
```

The `openaps * add` commands all change some of the INI configurations.

Did the **use** menu change at all? `openap use -h`

```
[...]
Known Devices Menu:
  These are the devices openaps knows about:

  device                Name and description:
    howdy               process - a fake vendor to run arbitrary commands
```

Now there's a **howdy** device in the **use** menu. The **use** menu adapts to our custom devices. Now we can **use** the device interactively: `openaps use howdy -h`, remember, we can always add `-h` to get more help/hints. Take note of that `--format text` option... our trivial `howdy` tool just prints a line of text. Most tools actually use a format called json (and it's the default), but for this example, we'll stick with `--format text`.

```
usage: openaps-use howdy [-h] USAGE ...

optional arguments:
  -h, --help  show this help message and exit

## Device howdy:
  vendor openaps.vendors.process

  process - a fake vendor to run arbitrary commands



  USAGE       Usage Details
    shell     run a process in a subshell
```

Hmm, because this a *unix process*, the **use** for this one is called **shell**. What happens if we just add that word to the end? `openaps use --format text howdy shell`

```
$ openaps use --format text howdy shell
hello world!
```

Now it prints `hello world!` because we are interactively **using** the **device**. The **device** was configured through the `add` command, and saved in the INI as the `process` **vendor**. The `process` **vendor** only exposes a single **use**: the `shell` use allows us to run any unix process. The **device** commands configure the **uses**. The **use** menu adapts to the registered **devices**. We can interact with a device by using it.

#### Summary

Hopefully this illustrates the relationship between the openaps **device** and **use** tools. The `device` command allows bringing devices into your instance, and **use** allows interacting with them. Let's take a deeper look at this relationship looking at the other vendors that might share a closer relationship to diabetes.

**Medtronic vendor** The medtronic vendor is backed by decocare. **Decocare** supports all paradigm series pumps.

Different pumps in the paradigm series have different features, decocare is aware of many of these differences but not all.

Adding a medtronic device to openaps also requires the **SERIAL** number of the pump. This is a six digit number, it's printed on the back of the pump. On the bottom right, there is a bar code, and right above that is the text: `SN PAR123456U`. In this case, the serial number is `123456`. It's also on the escape/status screen, scroll down, below the date, it will say: `S/N# 123456`, again the serial number would be `123456`.

For example purposes, we'll use the serial number `123456` here, you should use your pump's serial number. Also, for the purposes of this guide, we will not be issuing any commands that cause changes. For most of this tutorial, you do not need access to a medtronic pump, or even the carelink stick, this tutorial focuses on understanding how **devices** are related to **uses**.

**Configuring medtronic device** Let's add a `medtronic` **device** *named* `pump`. Remember that adding a device enables us to **use** it, and there's a `medtronic` vendor. `openaps device add pump medtronic -h` This one works a little different, it wants the serial number after:

```
usage: openaps-device add name medtronic [-h] serial

Medtronic - openaps driver for Medtronic

positional arguments:
  serial

optional arguments:
  -h, --help  show this help message and exit
```

So adding the serial to the end:

```
$ openaps device add pump medtronic 123456
added medtronic://pump
```

**openaps use pump** Based on our prior experience knowing that `device` enables **use**, let's check out our own `openap use -h` to see how it's changed.

```
[...]
Known Devices Menu:
  These are the devices openaps knows about:
```

```
  device                Name and description:
    howdy               process - a fake vendor to run arbitrary commands
    pump                Medtronic - openaps driver for Medtronic

Once a device is registered in openaps.ini, it can be used.
```

Now there's a **pump** device in the **use** menu! What can it do `openaps use pump -h`

```
usage: openaps-use pump [-h] USAGE ...

optional arguments:
  -h, --help            show this help message and exit

## Device pump:
  vendor openaps.vendors.medtronic

  Medtronic - openaps driver for Medtronic



  USAGE                 Usage Details
    Session             session for pump
    bolus               Send bolus command. [#warning!!!]
    filter_glucose_date
                        Search for glucose pages including begin and end dates
                        (iso 8601).
    filter_isig_date    Search for isig pages including begin and end dates
                        (iso 8601).
    iter_glucose        Read latest 100 glucose records
    iter_glucose_hours  Read latest n hours of glucose data
    iter_pump           Read latest 100 pump records
    iter_pump_hours     Read latest n hours of pump records
    model               Get model number [#oref0] [#recommended] [#safe]
    mytest              Testing read_settings
    read_basal_profile_A
                        Read basal profile A.
    read_basal_profile_B
                        Read basal profile B.
    read_basal_profile_std
                        Read default basal profile.
    read_battery_status
                        Check battery status. [#oref0]
    read_bg_targets     Read bg targets. [#oref0]
    read_carb_ratios    Read carb_ratios. [#oref0]
    read_clock          Read date/time of pump [#oref0]
    read_current_glucose_pages
                        Read current glucose pages.
    read_current_history_pages
                        Read current history pages.
    read_glucose_data   Read pump glucose page
    read_history_data   Read pump history page
    read_insulin_sensitivies
                        XXX: Deprecated. Don't use. Use
                        read_insulin_sensitivities instead.
    read_insulin_sensitivities
                        Read insulin sensitivities. [#oref0]
    read_selected_basal_profile
                        Fetch the currently selected basal profile. [#oref0]
```

```
    read_settings       Read settings. [#oref0]
    read_status         Get pump status
    read_temp_basal     Read temporary basal rates. [#oref0]
    reservoir           Get pump remaining insulin
    resume_pump         resume pumping.
    scan                scan for usb stick
    set_clock           Set clock.
    set_temp_basal      Set temporary basal rates. [#oref0]
    settings            Get pump settings
    status              Get pump status (alias for read_status)
    suspend_pump        Suspend pumping.
```

This is a list of all things the `medtronic` **vendor** knows how to do. These things have all been exposed in `openaps` as a single `use` in a **uniform** and **reproducible** way. Even though our `howdy` example device was completely different, the way we **use** it is identical to the `pump` device.

Get some of the help for how to use a pump:

- `openaps use pump model -h`
- `openaps use pump reservoir -h`
- `openaps use pump read_clock -h`
- `openaps use pump iter_pump_hours -h`

**Understanding how to talk to medtronic** This portion is the first portion of the tutorial where we will actually talk to the pump using the carelink usb stick. Everything prior to this has been configuration. Most of the time, Medtronic's wireless interface is off. There's a special command that enables wireless communication for several minutes, this is configurable in using the `minutes` parameter in the `pump.ini` **extra** ini. The medtronic vendor here tracks whether or not the session is expired, and renews it before continuing, this RF initialization could take an extra 30 seconds.

So, how fast does `openaps use pump model` take the first time? How long does it take if you repeat it several times?

Try the following commands, these are all safe, read-only commands:

- `openaps use pump model`
- `openaps use pump reservoir`
- `openaps use pump read_clock`
- `openaps use pump iter_pump_hours 2`
- If you are using Medtronic for CGM, try `openaps use pump iter_glucose_hours 2`

If you are going to build tools to improve diabetes therapy, which pieces of data might you need to gather? You can interactively **use** each of these features to view and inspect the data.

### Reports - the value of reproducibility

Reports can automate this by recording the output of any **use** to a file. Reports make **uses reproducible**. Let's peek at `openaps report -h`.

```
usage: openaps-report [-h] [--version] {add,remove,show,invoke} ...

 openaps-report - configure reports

optional arguments:
```

```
  -h, --help            show this help message and exit
  --version             show program's version number and exit

## Reports Menu:
   reports - manage report configurations

  {add,remove,show,invoke}
                        Operation
    add                 add - add a new report configuration
    remove              remove - remove a device configuration
    show                show - show all reports
    invoke              invoke - generate a report

Manage which devices produce which reports.

Example workflow:

Use the add, remove, show to manage which reports openaps knows about.

  The add command adds a new report to the system.
  The syntax is: add <name> <reporter> <device> <use>

    openaps report add my-results.json json pump basals

    This example registers a json output, using the pump basals command, and
    stores the result in my-results.json.

  The show command will list or give more details about the reports registered with openaps.
  The syntax is: show [name]. The default name is '*' which should list all available reports.

    openaps report show

  The remove command removes the previously configured report from openaps.
  The syntax is: remove <name>

    openaps report remove my-results.json
    This example removes the report "my-results.json" from the openaps
    environment.

  openaps report invoke basals
                 <action> <name>
```

The **use** commands allow us to interact with devices. This is very useful for exploring and learning about what the devices can do, but the `openaps use` tool is little unwieldy if we put our commands in a script.

For example, consider a simple script which saves the output from our `howdy` device to a file called `howdy.txt`:

```
openaps use --format text howdy shell > howdy.txt
```

Now, if there were any tools that needed to use the output from `howdy` can retrieve it from the `howdy.txt` file.

What if we take a more complicated example? Try running these commands:

```
echo {} | json -e "this.foo = 'first thing'"
echo {} | json -e "this.bar = 'second thing'"
echo {} | json -e "this.baz = 'baz thing'"
```

These three bash commands will `echo` slightly different `json` objects to the terminal. Let's create devices for each of these, and pretend each is some important piece of data, such as glucose data or pump history, or bg targets.

We'll iterate on the above to implement some fake data for this tutorial:

```
openaps device add fake-cgm \
  process bash -c '"echo {} | json -e '\''this.cgm_fake=\"fake-cgm\"'\'' "'
openaps device add fake-pump \
  process bash -c '"echo {} | json -e '\''this.pump_fake=\"fake-pump\"'\'' "'
openaps device add fake-oref0 \
  process bash -c '"echo {} | json -e '\''this.oref0_fake=\"fake-oref0\"'\'' "'
```

Let's assume we want to get data from all these devices in order to run an algorithm on the complete data set. The algorithm we want to run needs all the information from all three devices in a file. Just to review the uses:

```
Known Devices Menu:
  These are the devices openaps knows about:

  device                Name and description:
    fake-cgm            process - a fake vendor to run arbitrary commands
    fake-oref0          process - a fake vendor to run arbitrary commands
    fake-pump           process - a fake vendor to run arbitrary commands
    howdy               process - a fake vendor to run arbitrary commands
    pump                Medtronic - openaps driver for Medtronic
```

We can interact with the devices like this, but this just prints the information to the screen, and then it's gone.

```
openaps use  fake-cgm shell
openaps use  fake-pump shell
openaps use  fake-oref0 shell
```

If we wanted to get all the information into a file, we might have to do something like this:

```
openaps use fake-cgm shell > fake-cgm-data.txt
openaps use fake-pump shell > fake-pump-data.txt
openaps use fake-oref0 shell > fake-oref0-data.txt
```

Let's try to create reports for each of our fake device uses before: `openaps report add -h`:

```
usage: openaps-report add [-h] report {base,text,stdout,JSON} device ...

add    - add a new report configuration

positional arguments:
  report
  {base,text,stdout,JSON}

optional arguments:
  -h, --help            show this help message and exit

Known Devices Menu:
  These are the devices openaps knows about:

  device                Name and description:
    fake-cgm            process - a fake vendor to run arbitrary commands
    fake-oref0          process - a fake vendor to run arbitrary commands
    fake-pump           process - a fake vendor to run arbitrary commands
    howdy               process - a fake vendor to run arbitrary commands
    pump                Medtronic - openaps driver for Medtronic
```

Interesting, apparently the **reports** know about valid **devices** also. Let's configure a **report** to save the `fake-cgm shell` data into a file called `fake-cgm-data.txt`:

```
openaps report add fake-cgm-data.txt JSON fake-cgm shell
```

Notice how the `...` in `openaps report add <name> <format> ...` and `openaps use ...` are identical. This is a design feature to encourage iterating through interactive usage, and then saving the commands that work into the openaps configuration using the **add** comands.

#### openaps report add saves use configuration

What did this add command do?

```
$ openaps report add fake-cgm-data.txt JSON fake-cgm shell
added fake-cgm://JSON/shell/fake-cgm-data.txt
```

Let's get a list of all our known reports: `openaps report show`

```
fake-cgm://JSON/shell/fake-cgm-data.txt
```

Or let's see what happened behind the scenes with `git show`:

```
commit 25104f7cd4e56e6cb2ca630ce06f927046a669a3
Author: Ben West <bewest@gmail.com>
Date:   Sun Mar 27 18:53:11 2016 -0700

    openaps-report add fake-cgm-data.txt JSON fake-cgm shell

          TODO: better change descriptions
          /usr/local/bin/openaps-report add fake-cgm-data.txt JSON fake-cgm shell

diff --git a/openaps.ini b/openaps.ini
index 8b1dbeb..f55161a 100644
--- a/openaps.ini
+++ b/openaps.ini
@@ -18,3 +18,10 @@ extra = fake-pump.ini
 vendor = openaps.vendors.process
 extra = fake-oref0.ini

+[report "fake-cgm-data.txt"]
+device = fake-cgm
+remainder = []
+use = shell
+json_default = True
+reporter = JSON
+
```

It modified the openaps configuration with information that matches the **use** configuration. It did not run the use, and it did not create the file, it only saves the configuration.

Let's go ahead and add the others:

```
openaps report add howdy.txt text howdy shell
openaps report add fake-pump-data.txt JSON fake-pump shell
openaps report add fake-oref0-data.txt JSON fake-oref0 shell
```

Notice these add commands saves information about how were using openaps. This allows us to **reproduce** the same logic every time.

#### Show

Let's take another look at `openaps report show`:

```
fake-cgm://JSON/shell/fake-cgm-data.txt
howdy://text/shell/howdy.txt
fake-pump://JSON/shell/fake-pump-data.txt
fake-oref0://JSON/shell/fake-oref0-data.txt
```

#### Invoke

It's time to generate a bunch of reports. Instead of attempting to reproduce all the **uses** again with all the right flags and in the right ways, we can **invoke** the previously saved uses.

First, if we haven't run any reports before, our directory might look like this. There's no data here yet, just some configuration.

```
$ ls
fake-cgm.ini
fake-oref0.ini
fake-pump.ini
foo.ini
howdy.ini
openaps.ini
pump.ini
```

Let's try invoking a single report and see what happens:

```
$ openaps report invoke howdy.txt
howdy://text/shell/howdy.txt
reporting howdy.txt
$ ls
fake-cgm.ini
fake-oref0.ini
fake-pump.ini
foo.ini
howdy.ini
howdy.txt
openaps.ini
pump.ini
```

Now there's a `howdy.txt` file containing `cat howdy.txt`:

```
hello world!
```

#### Invoke runs preconfigured uses

`openaps report invoke` takes a list of any number of reports:

```
openaps report invoke \
  fake-cgm-data.txt howdy.txt fake-pump-data.txt fake-oref0-data.txt
```

```
$ openaps report \
  invoke fake-cgm-data.txt howdy.txt fake-pump-data.txt fake-oref0-data.txt
fake-cgm://JSON/shell/fake-cgm-data.txt
reporting fake-cgm-data.txt
```

```
howdy://text/shell/howdy.txt
reporting howdy.txt
fake-pump://JSON/shell/fake-pump-data.txt
reporting fake-pump-data.txt
fake-oref0://JSON/shell/fake-oref0-data.txt
reporting fake-oref0-data.txt
```

Now we can **invoke** many groups of reports in one line, save the data to their own files consistently, while referring to the preconfigured **use** for that **device**.

Each `invoke` creates a new git commit in the log: `openaps show`:

```
commit eb782e12552ad664697aa38d7e6b05b41f5e5a22
Author: Ben West <bewest@gmail.com>
Date:   Sun Mar 27 19:11:37 2016 -0700

    openaps-report invoke fake-cgm-data.txt howdy.txt fake-pump-data.txt fake-oref0-data.txt

          TODO: better change descriptions
          /usr/local/bin/openaps-report invoke fake-cgm-data.txt howdy.txt fake-pump-data.txt fake-or

diff --git a/fake-cgm-data.txt b/fake-cgm-data.txt
new file mode 100644
index 0000000..f54d1ff
--- /dev/null
+++ b/fake-cgm-data.txt
@@ -0,0 +1,3 @@
+{
+  "cgm_fake": "fake-cgm"
+}
\ No newline at end of file
diff --git a/fake-oref0-data.txt b/fake-oref0-data.txt
new file mode 100644
index 0000000..adb1295
--- /dev/null
+++ b/fake-oref0-data.txt
@@ -0,0 +1,3 @@
+{
+  "oref0_fake": "fake-oref0"
+}
\ No newline at end of file
diff --git a/fake-pump-data.txt b/fake-pump-data.txt
new file mode 100644
index 0000000..d2d5bf6
--- /dev/null
+++ b/fake-pump-data.txt
@@ -0,0 +1,3 @@
+{
+  "pump_fake": "fake-pump"
+}
\ No newline at end of file
```

Notice, we already ran `howdy`, earlier, and it did not change. Also notice how `invoke` performs the same exact logic for each **report** mentioned. It is equivalent to running the exact **use** for each command, saving the data in a file, and creating a log entry.

Reports are how we organize and track the data flowing through the system. In `openaps` **reports reproduce uses**.

### Alias - shortcut for any command

An **alias** allows us to assign a nickname to any command or group of commands. It's very similar to git alias, let's take a look at the `openaps alias --help` output:

```
usage: openaps-alias [-h] {add,remove,show} ...

 openaps-alias - manage aliases

optional arguments:
  -h, --help          show this help message and exit

## Alias Menu:
   aliases - manage alias configurations

  {add,remove,show}  Operation
    add              add - add an alias
    remove           remove - remove an alias
    show             show - show all aliases
```

Let's try a very trivial example with hello world again, `echo hello world` as an alias:

Adding an alias takes a name, and an alias definition (the commands to run). The commands to run may be any command inside `openaps` toolkit, or if it starts with a bang (`!`), it can run any arbitrary tool available on the system.

#### Hello world example

```
$ openaps alias add echo "! bash -c \"echo hello \$1\" --"
added echo ! bash -c "echo hello $1" --
$ openaps echo HUMAN
hello HUMAN
```

#### Openaps example

We can "rename" commands this way, for example we can alias `openaps invoke` to `openaps report invoke`:

```
$ openaps alias add invoke "report invoke"
added invoke report invoke
$ openaps invoke fake-cgm-data.txt fake-oref0-data.txt
fake-cgm://JSON/shell/fake-cgm-data.txt
reporting fake-cgm-data.txt
fake-oref0://JSON/shell/fake-oref0-data.txt
reporting fake-oref0-data.txt
```

#### Grouping commands logically

We can also group large groups of command invocations into one simple alias:

```
$ openaps alias add gather-all-fake \
  "report invoke howdy.txt fake-pump-data.txt fake-cgm-data.txt fake-oref0-data.txt"
added gather-all-fake report invoke howdy.txt fake-pump-data.txt fake-cgm-data.txt fake-oref0-data.tx
```

```
$ openaps gather-all-fake
howdy://text/shell/howdy.txt
reporting howdy.txt
fake-pump://JSON/shell/fake-pump-data.txt
reporting fake-pump-data.txt
fake-cgm://JSON/shell/fake-cgm-data.txt
reporting fake-cgm-data.txt
fake-oref0://JSON/shell/fake-oref0-data.txt
reporting fake-oref0-data.txt
```

An alias runs all the commands associated with it's definition. It's the same as running the commands themselves.

CHAPTER 3

# Walkthrough

## 3.1 Phase 0: General Setup

### 3.1.1 General Setup and Project Prep

The setup process is broken into four parts: acquiring the hardware you need; storing baseline data; configuring the Raspberry Pi and installing the openaps tools and dependencies. After completing these steps, you will be able to use the openaps tools to communicate with your insulin pump and CGM.

At this stage, you may want to begin documenting each step that you take. This will help in two ways.

First, this enables you to better ask for assistance if you run into errors, bugs, etc. By explaining where you are in the documentation and what you're seeing (by copying and pasting your last command and the output), someone can better provide tips on what you should consider next.

Second, this will enable you to help us improve our documentation. Did we skip a step, or not explain clearly? After you get through the setup instructions, you should consider forking a copy of these docs, editing with any changes you think should be made, and submitting a pull request (PR) back to the master. Others will be able to review & discuss any edits, make further changes, and pull this edits into the main file. This helps us all "pay it forward" as we go!

### 3.1.2 Baseline data

There is no requirement to share your data to use the openaps toolset or participate in the OpenAPS project. Individuals within the project who share their data do so at will and you should do the same only if you feel comfortable. That being said, it is always a good idea to record your data before embarking on a new set of experiments. This will be helpful to understand the effects of the system as well as gain a better understanding of your response to different control strategies.

#### CGM Data

Before getting started, we ask that you store at least 30 days of CGM data. For now, the easiest way to do that is to upload your Dexcom receiver to Dexcom Studio or, if you use a Medtronic CGM, upload your CGM data to CareLink. We suggest you get in the habit of doing this regularly so that you have ongoing data to show trends in your overall estimated average glucose (eAG, a good indicator in trends in A1c) and variations in your "time in range."

### Recent A1c

Go ahead and document your most recent A1c and keep it somewhere handy. This will allow you to compare your before/after results as well as be able to share it (if you choose) once there is a request from OpenAPS researchers, who may aggregate anonymous data to show what happens when people use OpenAPS.

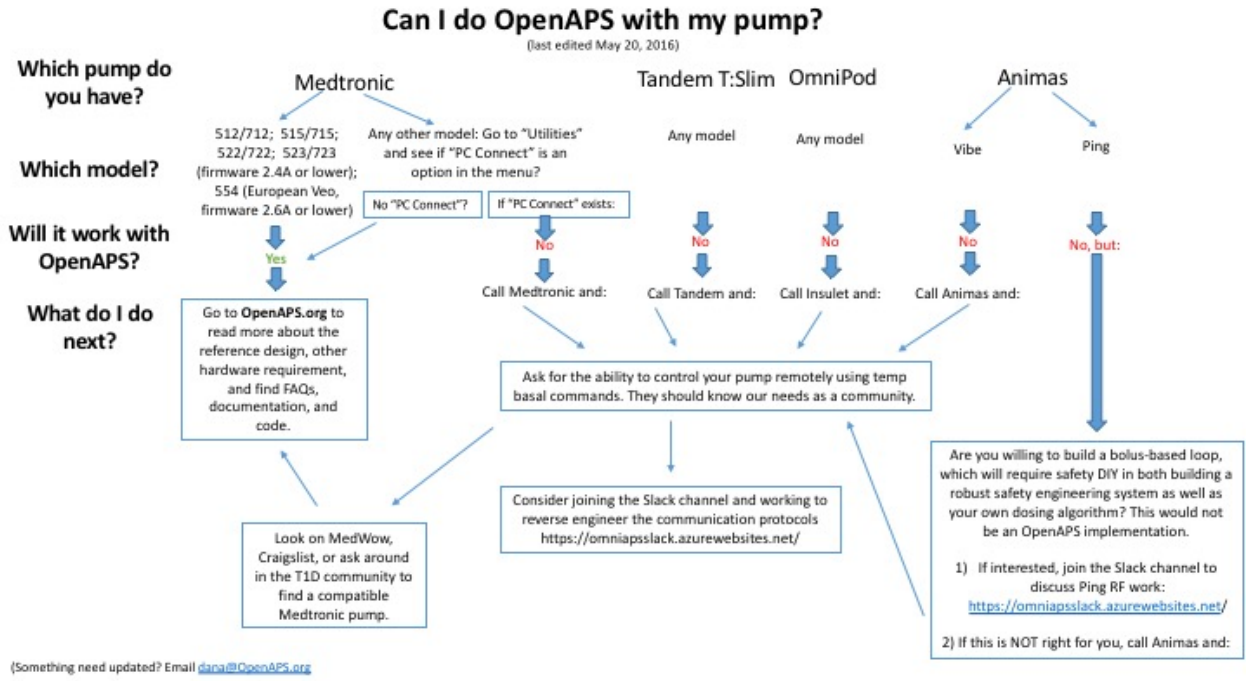
### Get comfortable making a PR

You should be comfortable making a PR to this documentation. If you haven't ever done this, take a look at my first pull request guide and try it out.

## 3.1.3 Hardware

This section describes the hardware components required for a 'typical' OpenAPS implementation. There are numerous variations and substitutions that can be made but the following items are recommended for getting started. If you come across something that doesn't seem to work, is no longer available, or if you have a notable alternative, feel free to edit this document with your suggestions.

To start, here is a high-level guide for understanding if your pump is compatible for OpenAPS:




If you're interested in working on communication for another pump (Omnipod, Animas, etc), click here to join the collaboration group focusing on alternative pump communication.

### Required Hardware for a "typical" setup

- An Insulin Pump:
  - One of the following Medtronic MiniMed model #s:
    - 512/712
    - 515/715

    * 522/722
    * 523/723 (with firmware 2.4A or lower)
    * 554/754 (European Veo, with firmware 2.6A or lower)
    * (To check firmware, hit Esc on the home screen and scroll all the way to the bottom. You can also go into the Utilities menu and look for a PC Connect option. If that is present, the pump will *not* work for looping. If it's absent, it should be able to receive temp basal commands.)

- A way to communicate with the pump:
  - Medtronic CareLink USB stick is the recommended option for your initial loop setup
  - **Note** that there are now other hardware options available to communicate with the pump. Some positives to an alternative include better range; some negatives include having to solder and the fact that they're not documented in this set of documentation yet. But if you're interested, check out some of the alternatives in the mmeowlink wiki.
- A Continuous Glucose Monitor (CGM):
- Dexcom CGM (G4 Platinum or Platinum with Share system); a G5 can be used but at this point requires additional work to be used.
- OR
- Medtronic CGM (MiniMed Paradigm REAL-Time Revel or Enlite)
- Other Supplies:
  - **Note** the below setup is what is used for the documentation; again see the mmeowlink wiki for some alternatives to the Raspberry Pi.
- Raspberry Pi 2 Model B ("RPi2")**(see note below)
- 8 GB (or greater) micro SD card
- Micro SD card to regular SD card converter [optional, but recommended so that you can use the micro SD card in a regular sized SD card drive]
- Low-profile USB WiFi adapter
- 2.1 Amp (or greater) USB power supply or battery
- Micro USB cable(s)
- AAA batteries (for pump)
- Case [optional]
- Cat5 or Cat6 Ethernet cable [optional]
- HDMI cable [optional, used for connecting the RPi2 to a screen for initial setup ease]
- USB Keyboard [optional, used to interact with the RPi2 via its own graphics interface on your TV screen]
- USB Mouse [optional, for the same purpose]

** Several #OpenAPS contributors recommend the Raspberry Pi 2 CanaKit, which includes several essential accessories in one package and can be purchased through Amazon

The CanaKit has the RPi2, SD card, WiFi adapter, and wall power supply. It also comes with a case, HDMI cable, and heat sink, none of which are required for an OpenAPS build. The kit does not have a micro USB cable (required to connect a Dexcom G4 receiver to the RPi) or a battery, which can be used in lieu of the wall power supply for portability.

Eventually, once you have an entire OpenAPS build up and running, it is recommended that you have backup sets of equipment in case of failure.

## Hardware Details & Recommendations

### Medtronic Insulin Pump

See currently known working list of pumps above. The easiest way to navigate to the Utilities / Connect Devices menu on your pump. If "PC Connect" is present in this menu, your pump is *not* compatible with OpenAPS.

Due to changes in the firmware, the openaps tools are only able to function in full on the above pump models. Security features were added in firmware version 2.5A that prevent making some remote adjustments via the CareLink USB stick. Each pump series is slightly different, and openaps functionality is still being ironed out for some of them. For 512/712 pumps, certain commands like Read Settings, BG Targets and certain Read Basal Profile are not available, and requires creating a static json for needed info missing to successfully run the loop (see example here).

If you need to acquire an appropriate pump check CraigsList or other sites like Medwow or talk to friends in your local community to see if there are any old pumps lying around in their closets gathering dust. MedWow is an eBay-like source for used pumps. Note: If you're buying a pump online, we recommended you ask the seller to confirm the firmware version of the pump. (You may also want to consider asking for a video of the pump with working functionality before purchasing.)

There are several #OpenAPS participants working on ways to use other pumps (including non-Medtronic models). If you would like to get more information on the progress in these areas, take a look at the #OpenAPS Google Group or click here to join the Slack channel.

### CareLink USB Stick

Currently, the primary supported device* (in the openaps documentation) for uploading pump data and interfacing on the #OpenAPS is the CareLink USB stick. We recommend you purchase at least two sticks because if one breaks, acquiring another stick will take time and will delay development. Additionally, due to the short range of communication between the CareLink stick and the Medtronic pumps, some users set up multiple sticks in different locations to maximize the chances of successful transmissions.

Some places to purchase: Medtronic or American Diabetes Wholesale.

A limitation of the Carelink USB stick is the short range of radio communications with the Medtronic pump. The radio signals are trasmitted from the end of the stick opposite the USB connector, on the flat grey side of the stick (see this set of experiments for details). Using a USB extension cable and angling the stick appropriately will assist in improving the connection.

Rerii 90 Degree USB Extension Cable

Mediabridge Products USB Extension Cable

### CGM: Dexcom G4 Platinum System (with or without Share) OR Medtronic

The openaps tool set currently supports two different CGM systems: the Dexcom G4 Platinum system (with or without the Share functionality) and the Medtronic system. With Dexcom, the Share platform is not required as communication with the receiver is usually accomplished via USB directly to the Pi. A G5 can also be used, but may require some extra work beyond this setup guide in order to configure. You can also [pull CGM data from Nightscout as an alternative, or use xDrip (see below). The Medtronic CGM system communicates directly with the associated pump, so the data can be retrieved using the CareLink USB stick.

Using the Dexcom CGM:

Note: Your Dexcom should be nearly fully charged before plugging it in to your Raspberry Pi. If, when you plug in your receiver, it causes your WiFi dongle to stop blinking, that is a sign that it is drawing too much power and needs to be charged. Once the receiver is fully charged, it will stay charged when connected to the Pi.

Your OpenAPS implementation can also pull CGM data from a Nightscout site in addition to pulling from the CGM directly.

- You can find more documentation about pulling CGM data from a Nightscout site here.
- If you have an Android phone, you can use the xDrip app to get your data from the Dexcom to Nightscout, to then be used in OpenAPS.
- If you have a Share receiver follow these directions to set up your Android uploader and Nightscout website.
- You could also build a DIY receiver. Directions to build the receiver, set up your uploader and Nightscout can be found here.
- You can also use part of the DIY receiver set up - the wixel – directly to the raspberry pi. Learn more about the wixel setup here and here.

Using the Medtronic CGM:

Because the Medronic pump collects data directly from the Enlite sensors, OpenAPS will retrieve CGM data in addition to your regular pump data from your pump. While you use the same OpenAPS commands to get it, the Medtronic CGM data need a little special formatting after being retrieved. We'll discuss these special circumstances as they come up later.

#### Raspberry Pi 2 Model B

The Raspberry Pi 2 (RPi2) model B is a credit-card sized single-board computer. The RPi2 primarily uses Linux kernel based operating systems, which must be installed by the user onto a micro SD card for the RPi2 to work. The RPi2 currently only supports Ubuntu, Raspbian, OpenELEC, and RISC OS. We recommend installing either Ubuntu or Raspbian. In this tutorial, you will learn how to do a "cableless" and "headless" install of Raspbian. You will be able to access and control the RPi2 via an SSH client on Windows, Mac OS X, Linux, iOS, or Android.

The RPi2 has 4 USB ports, an ethernet port, an HDMI port, and a micro USB power-in jack that accepts 2.1 Amp power supplies. In this tutorial, you will need to access the USB ports, micro USB power-in jack, and possibly the Ethernet jack (if wireless failure occurs). You will not require the HDMI port or a monitor.

Raspberry Pi 2 Model B

#### Micro SD Card

An 8 or 16 GB micro SDHC card is recommended. Get one that is class-4 or greater and is a recognized name brand, such as SanDisk, Kingston, or Sony. A list of verified working hardware (including SD cards) can be found here.

SanDisk Ultra 16GB Ultra Micro SDHC UHS-I/Class 10 Card with Adapter

Sony 16GB Class 10 UHS-1 Micro SDHC

Note: A known issue with the Raspberry Pi is that the SD card may get corrupted with frequent power cycles, such as when the system gets plugged and unplugged frequently from an external battery. Most core developers of openaps recommend purchasing extra SD cards and having them pre-imaged and ready to use with a backup copy of openaps installed, so you can swap it out on the go for continued use of the system.

#### WiFi Adapter

A minimalistic, unobtrusive WiFi USB adapter is recommended. The low-profile helps to avoid damage to both the RPi2 and the adapter as the RPi2 will be transported everywhere with the user.

Edimax EW-7811Un 150Mbps 11n Wi-Fi USB Adapter

Buffalo AirStation N150 Wireless USB Adapter

#### 2.1 Amp USB Battery Power Supply

A large-capacity power supply that is greater than 8000 mAh (milliAmp-hours) is recommended for full day use. Make sure that the battery has at least one 2.1 Amp USB output. A battery with a form-factor that minimizes size is recommended, to allow the patient to be as ambulatory as possible. When you have a full OpenAPS implemented and working, you will want to have multiple batteries to rotate and recharge. A battery that can deliver power while it charges is ideal as you will be able to charge it on-the-fly without shutting down and restarting the RPi2.

TeckNet® POWER BANK 9000mAh USB External Battery Backup Pack

Zendure® 2nd Gen A3 Portable Charger 10000mAh - 2.1a Dual USB - in-line charging

#### USB Cables

USB cables with a micro connector on one end and a standard (Type A) connector on the other are used to connect the power supply and the Dexcom receiver to the RPi2. Most cables will work fine, but some prefer to select lengths and/or features (such as right-angled connectors) to improve portability.

Rerii Black Golden Plated 15 cm Length Micro-B Male Left Angle USB cable

Monoprice Premium USB to Micro USB Charge, Sync Cable - 3ft

#### AAA Batteries

Repeated wireless communication with the pump drains the battery quite quickly. With a loop running every five minutes, a standard alkaline AAA—recommended by Medtronic—lasts somewhere between four to six days before the pump goes to a "Low Battery" state and stops allowing wireless transmission. Lithium batteries last significantly longer but do not give much warning when they are about to die, but alerts can be created to provide warning about the status of the battery. For further information on batteries, see this study on AAA battery use in a looping pump.

#### Cases

The Raspberry Pi is extremely minimalistic and does not come in a protective case. This is fine for development work, but presents an issue for day-to-day use. There are hundreds of cases available, but here is an example of what others are using in their OpenAPS builds.

JBtek® Jet Black Case for Raspberry Pi B+ & Raspberry Pi 2 Model B

Additionally, for mobile use, it is helpful to have something besides a lunchbox to carry the entire rig around. The size and weight of the component set as well as the limited range of the CareLink USB stick constrains the options here, but there are still some workable solutions. Waist-worn running gear and camera cases seem to work well. Two options: FlipBelt and Lowepro Dashpoint 20.

### 3.1.4 Setting Up Your Raspberry Pi

**Note:** Settuping up a Raspberry Pi is not specific to OpenAPS. Therefore, it's very easy to Google and find other setup guides and tutorials to help with this process. This is also a good way to get comfortable with using Google if you're unfamiliar with some of the command line tools. Trust us - even if you're an experienced programmer, you'll be doing this throughout the setup process.

In order to use the RPi2 with openaps development tools, the RPi2 must have an operating system installed and be set up in a very specific way. There are two paths to the intial operating system instalation and WiFI setup. Path 1 is recommended for beginners that are very new to using command prompts or "terminal" on the Mac. Path 2 is considered the most convenient approach for those with more experience with coding and allows the RPi2 to be set up without the use of cables, which is also known as a headless install. Either path will work and the path you choose is a matter of personal preference. Either way, it is recommended that you purchase your RPi2 as a CanaKit, which includes everything you will need for a GUI install.

For the Path 1 GUI install you will need:

- A Raspberry Pi 2 CanaKit or similar, which includes several essential accessories in one package
- USB Keyboard
- USB Mouse
- A TV or other screen with HDMI input

For the Path 2 Headless install, you will need:

- Raspberry Pi 2
- 8 GB micro SD Card [and optional adapter so that you can plug in the micro SD Card into your computer]
- Low Profile USB WiFi Adapter
- 2.1 Amp USB Power Supply
- Micro USB cable
- Raspberry Pi 2 CanaKit
- Console cable, ethernet cable, or Windows/Linux PC that can write ext4 filesystems

#### Download and Install Raspbian Jessie

Note: If you ordered the recommended CanaKit, your SD card will already come imaged. However, if you don't already know whether it's Raspbian 8 Jessie or newer (*see below*), just treat it as a blank SD card and download and install the latest versian of Raspbian (currently version 8.0, codename Jessie).

##### Download Raspbian

Raspbian is the recommended operating system for OpenAPS. Download the latest version (Jessie September 2015 or newer) of Raspbian here. Make sure to extract the disk .img from the ZIP file. Note that the large size of the Raspbian Jessie image means its .zip file uses a different format internally, and the built-in unzipping tools in some versions of Windows and MacOS cannot handle it. The file can be successfully unzipped with [7-Zip] (http://www.7-zip.org/) on Windows and [The Unarchiver] (https://itunes.apple.com/us/app/the-unarchiver/id425424353?mt=12) on Mac (both are free).

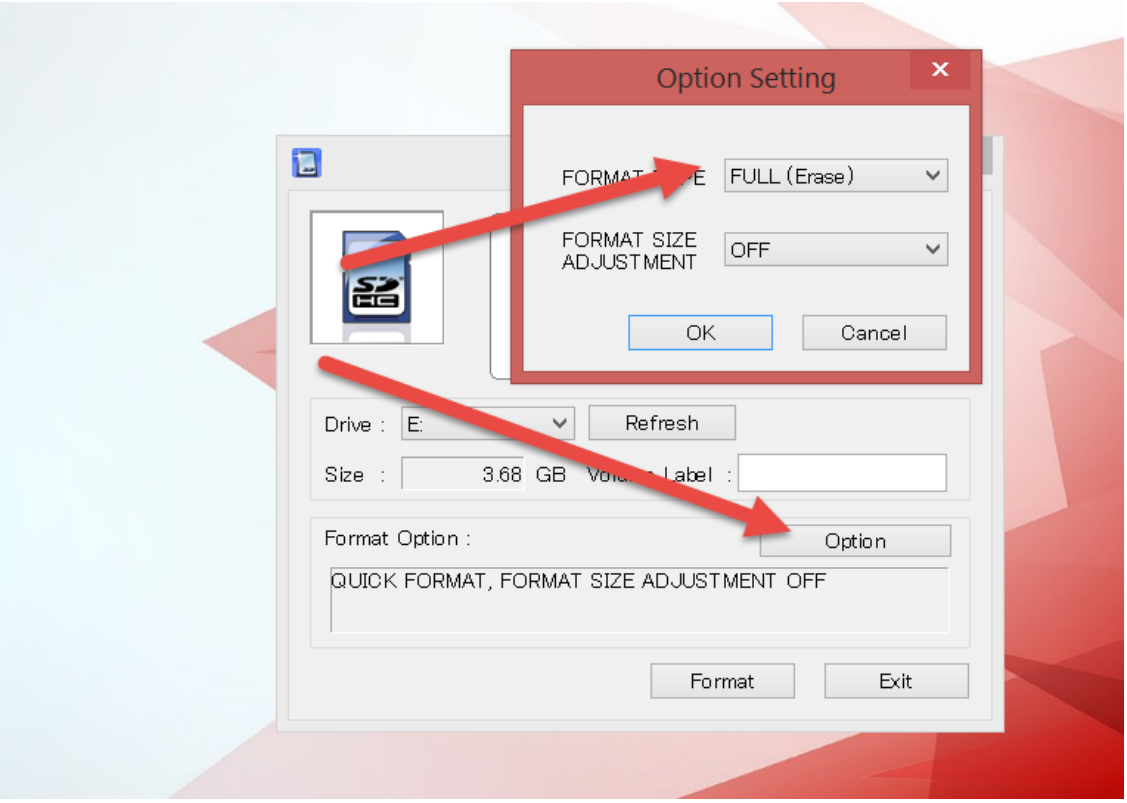
### Write Raspbian to the Micro SD Card

Erase (format) your SD card using https://www.sdcard.org/downloads/formatter_4/

Write the Raspbian .img you extracted from the ZIP file above to the SD card using the instructions at https://www.raspberrypi.org/documentation/installation/installing-images/

**Detailed Windows Instructions**

- First, format your card to take advantage of the full size it offers
  - If you got your through CanaKit, when you put it in your PC it will look like it is 1GB in size despite saying it is 8GB
- Download and install: https://www.sdcard.org/downloads/formatter_4/
- Run SDFormatter
  - Make sure your Micro SD Card is out of your Raspberry PI (shut it down first) and attached to your computer
  - Choose the drive where your card is and hit "Options"
  - Format Type: Change to Full (Erase)
  - This will erase your old Rasbian OS and make sure you are using the full SD card's available memory




  -
  - Format the card
- Download Rasbian 8 / Jessie
  - https://www.raspberrypi.org/downloads/raspbian/

  - Extract the IMG file
- Follow the instruction here to write the IMG to your SD card
  - https://www.raspberrypi.org/documentation/installation/installing-images/README.md
- After writing to the SD card, safely remove it from your computer and put it back into your RPi2 and power it up

### Connect and configure WiFi

- Insert the included USB WiFI into the RPi2.
- Next, insert the Micro SD Card into the RPi2.

#### Path 1: Keyboard, Mouse, and HDMI monitor/TV

- First, insert your USB keyboard and USB mouse into the RPi2.
- Next, connect your RPi2 to a monitor or T.V. using the included HDMI cable.
- Finally connect your RPi2 using the power adapter.
- You should see the GUI appear on sceen.
- Configure WiFi per the instruction pamphlet included with your CanaKit.
- Once you have installed Rasbian and connected to WiFI, you can disconnect the mouse, keyboard and HDMI cable.

Remember to keep your RPi2 plugged in, just disconnect the peripherals. Also remember to never disconnect your RPi2 without shutting it down properly using the `sudo shutdown -h now` command. If you are unable to access the Pi and must power it off without a shutdown, wait until the green light has stopped flashing (indicating the Pi is no longer writing to the SD card).

You can now skip to *Test SSH Access* and SSH into your RPi2.

#### Path 2: Console or Ethernet cable

- Get and connect a console cable (use this guide),
- Temporarily connect RPi to a router with an ethernet cable and SSH in (see below), or
- Connect the RPi directly to your computer with an ethernet cable (using this guide) and SSH in (see below)

**Configure WiFi Settings** Once you connect to the Pi, you'll want to set up your wifi network(s). It is recommended to add both your home wifi network and your phone's hotspot network if you want to use OpenAPS on the go.

To configure wifi:

Type `sudo bash` and hit enter

Input `wpa_passphrase "<my_SSID_hotspot>" "<my_hotspot_password>" >> /etc/wpa_supplicant/wpa_supplicant.conf` and hit enter (where `<my_SSID_hotspot>` is the name of your phone's hotspot and `<my_hotspot_password>` is the password).

(It should look like: `wpa_passphrase "OpenAPS hotspot" "123loveOpenAPS4ever" >> /etc/wpa_supplicant/wpa_supplicant.conf`)

Input your home wifi next: `wpa_passphrase "<my_SSID_home>" "<my_home_network_password>" >> /etc/wpa_supplicant/wpa_supplicant.conf` (and hit enter)

You can now skip to *Test SSH Access* and SSH into your RPi2.

#### Path 3: Headless WiFi configuration (Windows/Linux only)

Keep the SD card in the reader in your computer. In this step, the WiFi interface is going to be configured in Raspbian, so that we can SSH in to the RPi2 and access the device remotely, such as on a computer or a mobile device via an SSH client, via the WiFi connection that we configure. Go to the directory where your SD card is with all of the files for running Raspbian on your RPi2, and open this file in a text editor.

`/path/to/sd/card/etc/wpa_supplicant/wpa_supplicant.conf`

In this file you will list your known WiFi networks so your Pi can connect automatically when roaming (e.g., between your home WiFi and your mobile hotspot).

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="YOURMOBILESSID"
        psk="YOURMOBILEPASS"
}
network={
        ssid="YOURHOMESSID"
        psk="YOURHOMEPASS"
}
```

You can add as many network as you need, the next reboot your system will connect to the first available network listed in your config files. Once the network to which your board is connected becomes unavailable, it start looking for any other known network in the area, and it connects to it if available.

If you want to connect to a router which doesn't broadcast an SSID, add a line with `scan_ssid=1` after the `ssid` and `psk` lines for that network. (More info and examples for the options you can specify for each network are here.)

Boot your Pi. (Put the SD card into the RPi2. Plug in the compatible USB WiFi adapter into a RPi2 USB port. Get a micro USB cable and plug the micro USB end into the side of the RPi2 and plug the USB side into the USB power supply.)

If you are unable to access this file on your computer:

- Connect your Pi to your computer with an ethernet cable and boot your Pi
- Log in using PuTTY. The Host Name is `raspberrypi.local` and the Port is 22. The login is `pi` and the password is `raspberry`.
- Type `sudo nano /etc/wpa_supplicant/wpa_supplicant.conf` and edit the file as described above.

### Test SSH Access

#### Windows

Make sure that the computer is connected to the same WiFi router that the RPi2 is using. Download PuTTY here. Hostname is `pi@raspberrypi.local` and default password for the user `pi` is `raspberry`. The port should be set to 22 (by default), and the connection type should be set to SSH. Click `Open` to initiate the SSH session.

#### Mac OS X / Linux

Make sure that the computer is connected to the same WiFi router that the RPi2 is using.

Open Terminal and enter this command:

```
ssh pi@raspberrypi.local
```

Default password for the user `pi` is `raspberry`

#### iOS

Make sure that the iOS device is connected to the same WiFi network that the RPi2 is using. Download Serverauditor or Prompt 2 (use this if you have a visual impairment). Hostname is `pi@raspberrypi.local` and the default password for the user `pi` is `raspberry`. The port should be set to 22 (by default), and the connection type should be set to SSH.

You probably also want to make your phone a hotspot and configure the WiFi connection (as above) to use the hotspot.

#### Android

Make sure that the Android device is connected to the same WiFi network that the RPi2 is using. Download an SSH client in the Google Play store. Hostname is `pi@raspberrypi.local` and the default password for the user `pi` is `raspberry`. The port should be set to 22 (by default), and the connection type should be set to SSH. You may need to ssh using the ip address instead; the app "Fing - Network Tools" will tell you what the address is if needed.

You probably also want to make your phone a hotspot and configure the WiFi connection (as above) to use the hotspot.

Note: If connecting to the RPi2 fails at this point, the easiest alternative is to temporarily connect RPi to your router with an ethernet cable and SSH in, making sure both the computer and the RPi2 are connected to the same router.

### Configure the Raspberry Pi

#### Verify your Raspian Version

- In order to do this, you must have done Path 1 or Path 2 above so that you have an environment to interact with
- Go to the shell / Terminal prompt. If running the GUI, look at the Menu in the upper left and click the icon three to the right of it (looks like a computer)
- Type `lsb_release -a`
- If it says anything about Release 8 / Jessie, you have the correct version and can continue.
- If it says anything else, you need to go back to *Download and Install Raspbian Jessie*

#### Run raspi-config

Run

```
sudo raspi-config
```

Here you can expand filesystem to maximize memory, change user password and set timezone (in internationalization options). This will take effect on the next reboot, so go ahead and reboot if prompted, or run `sudo reboot` when you're ready.

Confirm that your keyboard settings are correct. Click on Menu (upper left corner of the screen, with raspberry icon). Mouse down to Preferences, and over to Mouse and Keyboard Settings. Click on Mouse and Keyboard Settings, then click on the Keyboard tab. Click on Keyboard Layout and be sure your country and variant are correct. For the US, it should be United States and English (US).

### Setting up an SSH key for Password-less Login [optional]

You can setup a public/private key identity, and configure your local computer and the Raspberry Pi to automatically use it. This will allow SSH access to the Pi without requiring a password. Some people find this feature very convenient.

#### Windows

If you don't already have an SSH key, follow this guide from GitHub to create one.

Create a .ssh directory on the Pi: run `mkdir .ssh`

Log out by typing `exit`

and copy your public SSH key into your RPi2 by entering

```
ssh-copy-id pi@raspberrypi.local
```

Now you should be able to log in without a password. Try to SSH into the RPi2 again, this time without a password.

#### Mac and Linux

If you don't already have an ssh key, then on your local computer (*not* on the Pi), run `ssh-keygen` (keep hitting enter to accept all the defaults).

If you created a new key identity and accepted all of the defaults, then the name of the newly generated identity will be `id_rsa`. However, if you set a custom name for the new identity (e.g. `id_mypi`), then you will need to add it to your local ssh keyring, via `ssh-add ~/.ssh/id_mypi`.

Next create a .ssh directory on the Pi: `ssh pi@raspberrypi.local`, enter the password for the `pi` user on the Pi, and run `mkdir .ssh`.

Next, add your new identity to the list of identities for which the Pi's `pi` user grants access via ssh:

```
cat ~/.ssh/<id_name>.pub | ssh pi@raspberrypi.local 'cat >>
.ssh/authorized_keys'
```

Instead of appending it to the list of authorized keys, you may simply copy your public key to the Pi, **overwriting its existing list of authorized keys**: `scp ~/.ssh/<id_name>.pub pi@raspberrypi.local:~/.ssh/authorized_keys`

Finally, `ssh pi@raspberrypi.local` to make sure you can log in without a password.

#### Disabling password login [optional - WARNING: THIS COULD POTENTIALLY LOCK YOU OUT OF YOUR RASPBERRY PI]

Make sure you fully understand this feature before proceeding. It is completely optional. There are two ways of securing the Pi:

1. Setting a password:

   a) use `sudo raspi-config` as described above, or

   b) `sudo passwd`

2. Disabling password login completely. In this case, you can ONLY log in with your SSH key. Be careful here.

   a) Open the `sshd_config` file in nano text editor on the Pi as follows

```
sudo nano /etc/ssh/sshd_config
```

```
b) Change the following
```

```
PermitRootLogin yes
# PasswordAuthentication yes
```

to

```
PermitRootLogin no
PasswordAuthentication no
```

Note that the second line was previously commented out.

From now on you will be able to SSH in with your private SSH key ONLY.

### Wifi reliability tweaks [optional]

Many people have reported power issues with the 8192cu wireless chip found in many wifi adapters when used with the Raspberry Pi. As a workaround, we can disable the power management features (which this chip doesn't have anyway) as follows:

```
sudo bash -c 'echo "options 8192cu rtw_power_mgnt=0 rtw_enusbss=0" >>
/etc/modprobe.d/8192cu.conf'
```

### Watchdog [optional]

Now you can consider installing watchdog, which restarts the RPi2 if it becomes unresponsive.

Enable the built-in hardware watchdog chip on the Raspberry Pi:

```
sudo modprobe bcm2708_wdog
```

```
sudo bash -c 'echo "bcm2708_wdog" >> /etc/modules'
```

Install the watchdog package, which controls the conditions under which the hardware watchdog restarts the Pi:

```
sudo apt-get install watchdog
```

Next, add watchdog to startup applications:

```
sudo update-rc.d watchdog defaults
```

Edit the config file by opening up nano text editor

```
sudo nano /etc/watchdog.conf
```

Uncomment the following: (remove the # from the following lines, scroll down as needed to find them):

```
max-load-1
watchdog-device
```

Finally, start watchdog by entering:

```
sudo service watchdog start
```

### Update the Raspberry Pi [optional]

Update the RPi2.

```
sudo apt-get update && sudo apt-get -y upgrade
```

The packages will take some time to install.

### Configure Bluetooth Low Energy tethering [optional]

The Raspberry Pi can be tethered to a smartphone and share the phone's internet connection. Bluetooth tethering needs to be enabled and configured on the phone device and your carrier/plan must allow tethering. The Raspberry Pi 3 has an inbuilt Bluetooth Low Energy (BLE) chip, while a BLE USB dongle can be used with the other Pi models.

The main advantages of using BLE tethering are that it consumes less power on the phone device than running a portable WiFi hotspot and it allows the Raspberry Pi to use whatever data connection is available on the phone at any given time - e.g. 3G/4G or WiFi. Some have also found that power consumption on the Raspberry Pi is lower when using BLE tethering compared to using a WiFi connection, although this may vary depending on BLE USB dongle, WiFi dongle, etc.

First, we clone a repository which contains scripts which are used later in the setup -

```
cd /home/pi
git clone https://github.com/WayneKeenan/RaspberryPi_BTPAN_AutoConnect.git
```

We then copy the required scripts into a 'bin' directory -

```
mkdir -p /home/pi/bin
cp /home/pi/RaspberryPi_BTPAN_AutoConnect/bt-pan /home/pi/bin
cp /home/pi/RaspberryPi_BTPAN_AutoConnect/check-and-connect-bt-pan.sh /home/pi/bin
```

To configure a connection from the command line -

```
sudo bluetoothctl
```

Enter the following commands to bring up the adapter and make it discoverable -

```
power on
discoverable on
agent on
default-agent
```

The adapter is now discoverable for three minutes. Search for bluetooth devices on your phone and initiate pairing. The process varies depending on the phone and the dongle in use. The phone may provide a random PIN and bluetoothctl may ask you to confirm it. Enter 'yes'. Then click 'pair' on the phone. Instead, the phone may ask you to enter a PIN. If so, enter '0000' and when bluetoothctl asks for a PIN, enter the same code again. Either way, bluetoothctl should inform you that pairing was successful. It will then ask you to authorize the connection - enter 'yes'.

Execute the paired-devices command to list the paired devices -

```
paired-devices
Device AA:BB:CC:DD:EE:FF Nexus 6P
```

Your paired phone should be listed (in this example, a Google Nexus 6P). Copy the bluetooth address listed for it; we will need to provide this later.

Now trust the mobile device (notice that bluetoothctl features auto-complete, so you can type the first few characters of the device's bluetooth address (which we copied previously) and hit to complete the address.

NOTE: Whenever you see 'AA:BB:CC:DD:EE:FF' or 'AA_BB_CC_DD_EE_FF' in this guide, replace it with the actual address of your mobile Bluetooth device, in the proper format (colons or underscores).

```
trust AA:BB:CC:DD:EE:FF
```

Quit bluetoothctl with 'quit'.

Now, we create a service so that a connection is established at startup. Execute the following commands to create a net-bnep-client.service file and open it for editing in Nano -

```
cd /etc/systemd/system
sudo nano net-bnep-client.service
```

In the editor, populate the file with the text below, replacing AA:BB:CC:DD:EE:FF with the address noted earlier -

```
[Unit]
After=bluetooth.service
PartOf=bluetooth.service

[Service]
ExecStart=/home/pi/bin/bt-pan client AA:BB:CC:DD:EE:FF

[Install]
WantedBy=bluetooth.target
```

Save the file, then enable the service -

```
sudo systemctl enable net-bnep-client.service
```

Open your crontab for editing -

```
crontab -e
```

...and add an entry to check the connection every minute and reconnect if necessary -

```
* * * * * /home/pi/bin/check-and-connect-bt-pan.sh
```

Save the file, then restart -

```
sudo shutdown -r now
```

### 3.1.5 Setting Up openaps and Dependencies

This section provides information on installing the base openaps toolkit and its dependencies.

#### Easy install of openaps and dependencies

##### Using the package manager

This is the recommended way to install:

```
curl -s https://raw.githubusercontent.com/openaps/docs/master/scripts/quick-packages.sh
| bash -
```

This uses this script to install all the dependencies in one step.

If the install was successful, the last line will say something like:

openaps 0.0.9 (although the version number may have been incremented)

If you do not see this or see error messages, try running the script multiple times.

#### Installing from source

It's possible to use the package manager to install development branches. If you are hacking on the code, you'll need a way to develop using versions you control. Here's a quick way to do that:

```
curl -s https://raw.githubusercontent.com/openaps/docs/master/scripts/quick-src.sh
| bash -
```

If successful, the last line will say something like:

openaps 0.0.10-dev (although the version number may have been incremented)

### Manual install [optional]

#### Install Python and Node.js Packages System-Wide [optional]

Run

```
sudo apt-get install python python-dev python-setuptools python-software-properties
python-numpy python-pip nodejs-legacy npm
```

This installs a number of packages required by openaps.

#### Install openaps [optional]

Run

```
sudo easy_install -ZU setuptools
```

```
sudo easy_install -ZU openaps
```

Running this command will also update openaps on your system if a newer version is available.

#### Install udev-rules [optional]

Run

```
sudo openaps-install-udev-rules
```

#### Enable Tab Completion [optional]

Run

```
sudo activate-global-python-argcomplete
```

#### Set up Git

Run

```
sudo apt-get install git
```

In order to set your git account's default identity, you will need to run the following two commands:

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

replace `you@example.com` and `Your Name` with your own information, but keep the quotes.

### 3.1.6 Loops In Progress

To get you comfortable with submitting a "PR" (stands for pull request), test it out by submitting a PR to this page, adding your name to the list of people who have loops in progress. This way we know how many people are in the development phase, too.

New to Github, and PRs? Check out how to submit your first PR.

List of people who are working on closed loops:

- Dana Lewis
- Ben West
- Chris Hannemann
- Sarah Howard
- Mike Stebbins
- Scott Hanselman
- Greg Scull
- Aaron Michelson
- Jayson EWER –Intel Edison w/ TI–cc1111
- Frank Best
- Brooke Armstrong & Matt Pazoles
- David Young
- Paul Martin
- Jarred Yaw
- Shane Mitchell
- Boris and Kayley Raskin
- Andy Pabari
- Rob Kresha - (Papillion, NE, USA)
- Christian Robinson (London, UK)
- Gary Kidd
- Nathan Morse
- Paul Davis (Brighton, UK)

## 3.2 Phase 1: Monitor

### 3.2.1 Phase 1: Logging, Cleaning, and Analyzing Your Data

Phase 1 focuses on accessing, logging, cleaning up, and analyzing data from the pump and CGM. Data fidelity is extremely important, especially when dosing is being considered. Take the time to review what the openaps tools are outputting and carefully compare the logs against your own CareLink and CGM reports.

By the end of this phase, you should have Nightscout set up; have alerts configured to know whether your system is looping or not; and be analyzing your existing pump or CGM data.

### 3.2.2 Configuring and Learning to Use openaps Tools

This section provides an introduction to intializing, configuring, and using the openaps toolset. The purpose is to get you familiar with how the different commands work and to get you thinking about how they may be used to build your own closed loop. Make sure you have completed the Setting Up the Raspberry Pi 2 and Setting Up openaps sections prior to starting.

The openaps readme has detailed information on the installation and usage of openaps. You should take the time to read through it in detail, even if it seems confusing at first. There are also a number of example uses available in the openaps-example repository.

Some familiarity with using the terminal will go a long way, so if you aren't comfortable with what `cd` and `ls` do, take a look at some of the Linux Shell / Terminal links on the Technical page.

Some conventions used in this guide:

- Wherever there are `<bracketed_components>` in the the code, these are meant for you to insert your own information. Most of the time, it doesn't matter what you choose **as long as you stay consistent throughout this guide**. That means if you choose `Barney` as your `< my_pump_name >`, you must use `Barney` every time you see `< my_pump_name >`. Choose carefully. Do not include the `< >` brackets in your name.

---

**Note:** One helpful thing to do before starting is to log your terminal session. This will allow you to go back and see what you did at a later date. This will also be immensely helpful if you request help from other OpenAPS contributors as you will be able to provide an entire history of the commands you used. To enable this, just run *$ script <filename>* at the beginning of your session. It will inform you that *Script started, file is <filename>*. When you are done, simply *$ exit* and it will announce *Script done, file is <filename>*. At that point, you can review the file as necessary.

---

#### Configuring openaps

##### Initialize a new openaps environment

To get started, SSH into your Raspberry Pi. Go to your home directory:

```
$ cd
```

Create a new instance of openaps in a new directory:

```
$ openaps init <my_openaps>
```

As mentioned above, `<my_openaps>` can be anything you'd like: `myopenaps`, `awesome-openaps`, `openaps4ever`, `bob`, etc.

Now that it has been created, move into the new openaps directory:

```
$ cd <my_openaps>
```

All subsequent openaps commands must be run in this directory. If you try to run an openaps command in a different directory, you will receive an error:

```
Not an openaps environment, run:  openaps init
```

The directory you just created and initialized as an openaps environment is mostly empty at the moment, as can been seen by running the list files command:

```
$ ls
openaps.ini
```

That `openaps.ini` file is the configuration file for this particular instance of openaps. It will contain all of your device information, plugin information, reports, and aliases. In the subsequent sections, you will be configuring your openaps instance to add these components. For now, however, it is blank. Go ahead and take a look:

```
$ cat openaps.ini
```

Didn't return much, did it? By the way, that `cat` command will be very useful as you go through these configuration steps to quickly check the contents of files (any files, not just `openaps.ini`). Similarly, if you see a command that you are unfamiliar with, such as `cat` or `cd`, Google it to understand what it does. The same goes for error messages—you are likely not the first one to encounter whatever error is holding you back.

#### Add openaps-contrib

This provides us with timezone support:

```
openaps vendor add openapscontrib.timezones
openaps device add tz timezones
```

### Get Pump data

#### Add pump as device

```
$ openaps device add <my_pump_name> medtronic
```

In order to communicate with the pump and cgm receiver, they must first be added as devices to the openaps configuration. To do this for a device we'll call `pump`:

```
$ openaps device add pump medtronic
```

Then to configure the new `pump` device with its serial number: `openaps use pump config --serial 123456`

Create bunch of reports: `oref0 template mint reports medtronic-pump | openaps import`

Here, `<my_pump_name>` can be whatever you like, but `<my_serial_number>` must be the 6-digit serial number of your pump. You can find this either on the back of the pump or near the bottom of the pump's status screen, accessed by hitting the ESC key.

**Important:** Be careful when choosing to share your 6-digit pump serial number or posting it online. If someone had access to this number, and was in radio reach of your pump, this could be used to communicate with your pump without your knowledge. While this is a feature when you want to build an OpenAPS, it is a flaw and a security issue if someone else can do this to you.

### Get CGM Data

#### From Dexcom CGM receiver via usb cable

Now you will do this for the Dexcom CGM receiver using a usb cable:

```
$ openaps device add cgm dexcom
```

Note this step is not required if you are using a Medtronic CGM. The pump serves as the receiver and all of the pumping and glucose functionality are contained in the same openaps device.

**G5 support** Support for G5 receiver is offered through the usb cable and configuring the `cgm` device with: `openaps use cgm config --G5`.

#### G4 with Share

Install BLE helpers:

```
sudo pip install git+git://github.com/bewest/Adafruit_Python_BluefruitLE.git'#wip/bewest/custom-gatt-
sudo pip install git+git://github.com/bewest/openxshareble.git
# adds openxshareble as vendor
openaps vendor add openxshareble
```

Then set up cgm device this way:

```
openaps device add cgm openxshareble
```

#### Glucose Data

Test ability to get data.

```
openaps use cgm oref0_glucose  --hours 2.0

# for G5 consider this instead:
openaps use cgm oref0_glucose  --no-raw --hours 2.0
```

Add as report:

```
openaps report add raw-cgm/glucose-raw.json JSON cgm oref0_glucose  --hours 2.0
# For G5:
openaps report add raw-cgm/glucose-raw.json JSON cgm oref0_glucose  --no-raw --hours 2.0
```

Ensure that the data is zoned correcty:

```
openaps use tz rezone  --date dateString --date display_time raw-cgm/glucose-raw.json
openaps  tz rezone  --date dateString --date display_time raw-cgm/glucose-raw.json
```

Save an alias to fetch CGM data:

```
openaps alias add monitor-cgm "report invoke raw-cgm/glucose-raw.json monitor/glucose.json"
```

Now, `openaps monitor-cgm` is available to pull in fresh CGM data from Dexcom.

#### Check that the devices are all added properly

```
$ openaps device show
```

should return something like:

```
medtronic://pump
dexcom://cgm
```

Here, `pump` was used for `<my_pump_name>` and `cgms` was used for `<my_dexcom_name>`. The names you selected should appear in their place.

Your `openaps.ini` file now has some content; go ahead and take another look:

```
$ cat openaps.ini
```

Now, both of your devices are in this configuration file:

```
[device "pump"]
vendor = openaps.vendors.medtronic
extra = pump.ini

[device "cgms"]
vendor = openaps.vendors.dexcom
extra = cgms.ini
```

Again, `pump` was used for `<my_pump_name>` and `cgms` was used for `<my_dexcom_name>`. Your pump model should also match your pump.

Because your pump's serial number also serves as its security key, that information is now stored in a separate ini file (here noted as `pump.ini`) that was created when you created the pump device. This makes it easier for sharing the `openaps.ini` file and also for keeping `pump.ini` and `cgms.ini` more secure. Be careful with these files. Open the pump's ini file now (use the name reported to you in the line labeled `extra` in the `openaps.ini` file).

```
$ cat pump.ini
```

It should show something like this:

```
[device "pump"]
serial = 123456
```

The serial number should match that of your pump.

If you made a mistake while adding your devices or simply don't like the name you used, you can go back and remove the devices as well. For example, to remove the pump:

```
$ openaps device remove <my_pump_name>
```

Then, you can add your pump again with a different name or serial number.

#### Check that you can communicate with your pump

Now that you have added these devices, let's see if we can establish communication with them. First, the pump:

```
$ openaps use <my_pump_name> model
```

should return something like:

```
"723"
```

Congratulations, you just pulled data from your pump! The `model` command is a very useful one to verify whether you can communicate with the pump. It is not, however, the only thing you can do. Take a look at the help file to see all of the possibilities:

```
$ openaps use <my_pump_name> -h
```

This returns a healthy bit of useful information, including a list of all the commands that can be done with `$ openaps use <my_pump_name>`. Of course, each one of those uses has its own help file as well:

```
$ openaps use <my_pump_name> model -h
usage: openaps-use pump model [-h]

 Get model number


optional arguments:
  -h, --help  show this help message and exit
```

The `-h` argument is your friend. If you ever forget what a command does, what arguments it requires, or what options it has, `-h` should be your first resource.

Go ahead and try some more pump uses to find out what they do. Note that some of the commands require additional inputs; these are detailed in the specific help files.

#### Check that you can communicate with your Dexcom receiver

Now let's try communicating with the Dexcom receiver.

Hint: Your Dexcom should be nearly fully charged before plugging it in to your Raspberry Pi. If, when you plug in your Dexcom, it causes your WiFi dongle to stop blinking, or if the charging icon on the Dexcom keeps cycling on and off, that is a sign that it is drawing too much power and needs to be charged.

> Workaround: If you continue to have problems, try increasing the mA output to the USB ports, you can do this by running the following command `$ sudo bash -c "echo -e \"#Enable Max USB power\nmax_usb_current=1\" >> /boot/config.txt"`.
>
> Reboot via `$ sudo shutdown -r now` to pick up the changes.

```
$ openaps use <my_dexcom_name> iter_glucose 1
```

should return something like:

```
[
  {
    "trend_arrow": "FLAT",
    "system_time": "2015-08-23T21:45:29",
    "display_time": "2015-08-23T13:46:21",
    "glucose": 137
  }
]
```

Hint: if this doesn't work, check to make sure that your Dexcom receiver is plugged into your Raspberry Pi ;-)

Just like with the pump, you can use the `-h` argument to call the help files. For example:

```
$ openaps use <my_dexcom_name> iter_glucose -h
usage: openaps-use cgms iter_glucose [-h] [count]

 read last <count> glucose records, default 100, eg:

positional arguments:
  count       Number of glucose records to read.

optional arguments:
```

```
  -h, --help  show this help message and exit

* iter_glucose   - read last 100 records
* iter_glucose 2 - read last 2 records
```

#### Pulling blood glucose levels from Nightscout

Some people have found it more beneficial to pull blood glucose values from Nightscout rather than directly from the Dexcom receiver.

Many people will actually setup both ways to pull the blood glucose level and switch between the different devices depending on their needs. If you are going to pull it directly from Nightscout then you will have to have internet access for the Raspberry Pi.

The `autoconfigure-device-crud` feature will allow us to create an easy to use `ns` device:

```
$ nightscout autoconfigure-device-crud https://my.nightscout.host averylongplainsecret
added process://ns/nightscout/ns NIGHTSCOUT_HOST API_SECRET
process://ns/nightscout/ns https://my.nightscout.host e6fc892e8e88235a476d197de3dfbef3f2db53d0
```

It added a new `ns` device to our uses menu:

```
openaps use ns shell get entries.json 'count=10'
openaps use ns shell upload treatments.json recently/combined-treatments.json
```

So we now have various uses for `ns`: **get**, **upload**, **latest-treatment-time**, **format-recent-history-treatments**, **upload-non-empty-treatments**.

**nightscout tools in openaps**

```
openaps use ns shell get entries.json 'count=10'
openaps use ns shell upload treatments.json recently/combined-treatments.json
```

```
  -h                                  This message.
  get type args                                 Get records of type from
                                                Nightscout matching args.

  upload endpoint file                          Upload a file to the Nightscout endpoint.
  latest-treatment-time                         - get latest treatment time from Nightscout
  format-recent-history-treatments history model - Formats medtronic pump
                                                history and model into
                                                Nightscout compatible
                                                treatments.

  format-recent-type ZONE type file             - Selects elements from the
                                                file where the elements would
                                                satisfy a gap in the last 1000
                                                Nightscout records.

  upload-non-empty-treatments file              - Upload a non empty treatments
                                                file to Nightscout.
  lsgaps tz entries                             - Re-use openaps timezone device
                                                to find gaps in a type (entries)
                                                by default.
  upload-non-empty-type type file
  status                                        - Retrieve status
  preflight                                     - NS preflight
```

### Nightscout Endpoints

- `entries.json` - Glucose values, mbgs, sensor data.
- `treatments.json` - Pump history, bolus, treatments, temp basals.
- `devicestatus.json` - Battery levels, reservoir.
- `profile.json` - Planned rates/settings/ratios/sensitivities.
- `status.json` - Server status.

### Examples

#### Get records from Nightscout

Use the get feature which takes two arguments: the name of the endpoint (entries, devicestatus, treatments, profiles) and any query arguments to append to the argument string. 'count=10' is a reasonable debugging value. The query-params can be used to generate any query Nightscout can respond to.

```
openaps use ns shell get $endpoint $query-params
```

#### Unifying pump treatments in Nightscout

To upload treatments data to Nightscout, prepare you zoned glucose, and pump model reports, and use the following two reports:

```
openaps report add nightscout/recent-treatments.json JSON ns shell  format-recent-history-treatments
openaps report add nightscout/uploaded.json JSON  ns shell upload-non-empty-treatments  nightscout/re
```

Here are the equivalent uses:

```
openaps use ns shell format-recent-history-treatments monitor/pump-history.json model.json
openaps use ns shell upload-non-empty-treatments nightscout/recent-treatments.json
```

The first report runs the format-recent-history-treatments use, which fetches data from Nightscout and determines which of the latest deltas from openaps need to be sent. The second one uses the upload-non-empty-treatments use to upload treatments to Nightscout, if there is any data to upload.

#### Uploading glucose values to Nightscout

Format potential entries (glucose values) for Nightscout.

```
openaps use ns shell format-recent-type tz entries monitor/glucose.json  | json -a dateString | wc -l
# Add it as a report
openaps report add nightscout/recent-missing-entries.json JSON ns shell format-recent-type tz entries
# fetch data for first time
openaps report invoke nightscout/recent-missing-entries.json

# add report for uploading to NS
openaps report add nightscout/uploaded-entries.json JSON  ns shell upload entries.json nightscout/rec
# upload for fist time.
openaps report invoke nightscout/uploaded-entries.json
```

### Adding and Invoking Reports

At this point, you should be comfortable communicating with your pump and cgm receiver with the `openaps use` command. This is great for learning and for experimenting, but it lacks the ability to generate output files. You'll notice that running

```
$ openaps use <my_dexcom_name> iter_glucose 100
```

prints *a lot* of data to the terminal. It would be great to save that data somewhere so that it can be used for logging historical records, performing calculations, and verifying actions. That is what `report` does.

Generating reports involves two steps: adding the report structures to your openaps configuration and invoking the reports to produce the desired outcome.

#### Adding Reports

As an example, let's suppose you would like to gather the last four hours of records from your pump. With the `use` command, that would be:

```
$ openaps use <my_pump_name> iter_pump_hours 4
```

This dumps the past four hours of pump records directly to the terminal.

Now, let's add this as a report instead:

```
$ openaps report add last_four_pump_hours.json JSON <my_pump_name>
iter_pump_hours 4
```

If done correctly, the only thing returned in the terminal is:

```
added pump://JSON/iter_pump_hours/last_four_pump_hours.json
```

Let's take a closer look at each section. `openaps report add` is adding a report to your openaps configuation. The report name is `last_four_pump_hours.json`. The format of the report is `JSON`. The command that will be used to generate the report is `<my_pump_name> iter_pump_hours 4`. You will notice that this last section is identical to what was called above when you printed the output to the terminal window, except there it was done with the `use` command. The report is simply running that same command and writing the output to the file you specified in the format you specified.

Much like adding devices, this report configuration is saved to your `openaps.ini` file. You can view all of your reports there with `$ cat openaps.ini` or by using `$ openaps report show`. Similarly, you can remove reports with `$ openaps report remove <report_name>`.

#### Invoking Reports

Adding the report does not actually generate the output file. To do this, you need to invoke the report:

```
$ openaps report invoke last_four_pump_hours.json
```

Again, the terminal output will be minimal:

```
pump://JSON/iter_pump_hours/last_four_pump_hours.json
reporting last_four_pump_hours.json
```

This time, however, a new file was created. Check and see using `$ ls`; you should see a file called `last_four_pump_hours.json` in your directory. Take a look at the file with `$ cat last_four_pump_hours.json`. The file's contents should look very familiar—the same data that was printed to ther terminal window when you performed `$ openaps use <my_pump_name> iter_pump_hours 4`.

Each time you add a new report to your configuration, you should immediately invoke it and check the resulting file. This means **open the file and actually check to make sure the output is what you expect**. Don't assume that it worked just because you didn't see an error.

The reports you add are reusable—each time you would like new data, simply invoke the report again and it will overwrite the output file. If you would like to see when the file was last edited, use the command `$ ls -l`. This will help you make sure you are getting up-to-data data.

Go ahead and create (and check) some reports for the the commands you have been using the most.

### Aliases

Now that you have some reports added, you may notice that you end up calling some of them in combinations. For example, you might always want to get your updated pump records and your updated cgm records. To do that, you would normally run two commands each time:

```
$ openaps report invoke last_four_pump_hours.json
$ openaps report invoke last_four_cgm_hours.json
```

For this example, we assume that you have added a second report called `last_four_cgm_hours.json` that is similar to the `last_four_pump_hours.json` we walked through previously, except that it is using your `<my_dexcom_name>` device and the `iter_glucose_hours` command. Go ahead and do that so you can follow along.

Calling two sequential commands for each update is a bit annoying, but imagine calling five or ten. Luckily, openaps has a built-in way to group these commands: aliases. Aliases allow generation of single-word commands to invoke a series of reports. For this example, create an alias called `last_four_hours`:

```
$ openaps alias add last_four_hours "report invoke last_four_pump_hours.json
last_four_cgm_hours.json"
```

Go ahead and execute this command:

```
$ openaps last_four_hours
```

You will see that it invokes each of the reports you specified in the order you specified. It prints each step out to the terminal window, and you will find that the corresponding output files have been created.

Just like with devices and reports, the alias is now part of your openaps configuration. You can view all of your aliases with `$ cat openaps.ini` or by using `$ openaps alias show`. Similarly, you can remove aliases with `$ openaps alias remove <alias_name>`.

Aliases will invoke reports and execute logic and shell commands. Aliases are not limited to reports. They can be nested. For example a top level alias is $ openaps alias1 alias2. alias2 is for example: Run shell "program1 && something2 || alias3" (one and two or three). alias3 is for example: $ Openaps report invoke settings/settings.json settings/bg_targets.json settings/insulin_sensitivities.json settings/basal_profile.json settings/profile.json

### Putting the Pieces Together

Take a moment to consider putting these commands to work in the larger context of a closed-loop system. Components of that system that you might need to `add` and `invoke` would be recent glucose data, recent pump history, the time, battery status, pump settings, carb ratios, the current basal profile, insulin sensitivities, blood glucose targets, and the status of the pump.

Go ahead and add and invoke reports for these components of a future closed-loop system.

Are there groupings of these reports that you imagine would be called at the same time? For example, in a closed-loop setup, the pump settings, blood glucose targets, insulin sensitivities, the basal profile, and carb ratios would not need to be checked as often as the current pump status, battery status, clock, recent blood sugars, and recent pump history.

Take some time to create aliases for groups of reports that would be called at the same time and verify that they invoke the expected reports. Reports will execute the "use" command. The "use" command is -h anotated. To see the annotation use this command $ openaps use -h

#### Backing Up Your openaps Instance

There are numerous ways to back up your system, from making a copy of the entire SD card to copying over individual files. Here, we will discuss one method of using git to back up just the openaps instance you've created. Note that this will not back up the entire sysem (and all the work you did in Setting Up the Raspberry Pi 2 and Setting Up openaps), but it will enable you to skip all of the configuration steps above if something happens.

For this backup method, we will take advantage of the fact that your openaps instanace is a git repository. We won't go over git here, but take a look at the references on the Resources page to get familiar with the basics. Typically, we would do this backup using GitHub, since that is where most of the openaps repositories are located and you should already have an account. However, GitHub only provides free repositories if they are public, and since this repository has your `<my_pump_name>.ini` file—and thus your pump's serial number—in it, we want to make sure that is private. If you are comfortable with sharing your glucose and pump history publicly, you can make sure the secret .ini files remain so by creating a .gitignore file listing `<my_pump_name>.ini` (and any other .ini files with secret information). This will prevent git from uploading those files to GitHub, but will still allow you to backup and publicly share all your other configuration and data.

---

**Note:** Note Alternatively, you can [purchase a monthly GitHub plan](https://github.com/pricing) and then follow [these instructions](https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/) if you'd like to go with GitHub, or use a service like Bitbucket instead.

---

Bitbucket offers a similar service to GitHub, but permits users to create free private repositories. Go ahead and sign up and then create a repository. You can call it whatever you like, but make sure that on the "Create a new repository" setup page you leave the "This is a private repository" box checked. Once created, you will be directed to a "Repository setup" page. Under the "Command line" section, click on the "I have an existing project" option and follow the instructions.

Once you have completed this step, all of the files in your `<my_openaps>` directory will be saved in your online Bitbucket repository. Whenever you would like to update your backup, simply go into your `<my_openaps>` directory and `$ git push`. This process can be automated, but we'll save that for another day.

Github mkimg tool has the ability to create a full SD card image with the advantage to shrink it to its minimum size quite different from the windows Win32DiskImager which always creates an image which is as large as the card. Using Win32DiskImager it might not be possible to restore an image on a new card with the same size. Using Github mkimg tool instead for example a 32 GB size card can be reduced to approximately 1.9 GB.

### 3.2.3 Visualization and Monitoring

Nightscout is the recommended way to visualize your OpenAPS closed loop. Even if you don't choose to share your Nightscout instance with another person, it will be helpful for you to visualize what the loop is doing; what it's been doing; plus generate helpful reports for understanding your data and also allowing you to customize watchfaces with your OpenAPS data. This provides a visual alternative to SSHing into your raspberry Pi or loop system and looking through log files.

#### Nightscout Integration

The integration requires setting up Nightscout and making changes and additions to your OpenAPS implementation.

### Nightscout Setup

OpenAPS requires the latest (currently dev) version of Nightscout, which can be found here: https://github.com/nightscout/cgm-remote-monitor/tree/dev. If you are already using Nightscout you might have to update your repository. Just go to the https://github.com/nightscout/cgm-remote-monitor repository and look for "updating my version". Once you have completed these steps, log on to Azure or Heroku and disconnect the deployment source. Thereafter choose your cgm-remote-monitor github repository as source again. You should take the dev branch of this repository especially if you plan to use the advanced-meal-assist feature.

The steps discussed here are essentially the same for both Azure and Heroku users. Two configuration changes must be made to the Nightscout implementation:

- Add "openaps" (without the quotes) and, optionally, "pump" (without the quotes) to the list of plugins enabled, and
- Add a new configuration variable DEVICESTATUS_ADVANCED="true" (without the quotes)

For Azure users, here is what these configuration changes will look like (with just "openaps" added): . For Heroku users, exactly the same changes should be made on the Config Vars page. The optional "pump" plugin enables additional pump monitoring pill boxes. For example, assuming you have added "pump" to the list of enabled plugins, you may add a new configuration variable `PUMP_FIELDS="reservoir battery"` to display pump reservoir and battery status on the Nightscout page. The "pump" plugin offers a number of other options, as documented on the Nightscout readme.

Next, on your Nightscout website, go to the Settings (3 horizontal bars) in the upper right corner. At the very bottom of the Settings menu, in the "About" section, you may check the Nightscout version (e.g. version 0.9.0-dev). Just above is a list of Plugins available. OpenAPS should show up. Click the check box to enable. Similarly, in the case you've enabled the "pump" plugin, "Pump" should also show up in the list, and you may check the box to enable. You should now see the OpenAPS pill box (and any optional pump monitoring pill boxes) on the left side of the Nightscout page near the time. You may also want to graphically show the basal rates: select "Default" or "Icicle" from the "Render Basal" pull-down menu in the Settings.

### Configure Nightscout profile

You need to create a profile in your Nightscout site that contains the Timezone, Duration of Insulin Activity (DIA), Insulin to carb ratio (I:C), Insulin Sensitivity Factor (ISF), Carbs Activity / Absorption rate, Basal Rates and Target BG range.

These settings are not currently updated from the values stored in the pump. You will need to keep the Nightscout profile in sync with any changes you make in your pump.

To configure your profile, on your Nightscout website, go to the Settings (3 horizontal bars) in the upper right corner.Click on the Profile Editor button.Create a new profile (if you don't already have one) using the settings that match what you already have set up in your pump.Fill out all the profile fields and click save.

### Configuring and Uploading OpenAPS Status

**At this point in the docs I find it confusing as the next part dives straight into working inside of your openaps repo (or whatever the right term for it is). I would think before jumping straight to setting up the integration with Nightscout you would go over some basic `openaps use ns` type stuff, or even just doing `openaps init` for the first time. I see this stuff is in Phase 2 - Configuring and Learning to Use openaps Tools, so the next bit seems out of place if you're supposed to follow the phases in order**

Integration with Nightscout requires couple of changes to your OpenAPS implementation, which include:

- Adding a new `ns` device, and generating a new report `monitor/upload-status.json`, which consolidates the current OpenAPS status to be uploaded to Nightscout

Upon successful completion of these two steps, you will be able to see the current OpenAPS status by hovering over the OpenAPS pill box on your Nightscout page, as shown here, for example:

The `ns` is a virtual device in the oref0 system, which consolidates OpenAPS status info in a form suitable for upload to Nightscout. First, add the device:

The reports required to generate upload-status.json should look familiar. If you have not generated any of these required reports, you should set them up and make sure they all work. In particular, note that monitor/clock-zoned.json contains the current pump clock time stamp, but with the timezone info included. If you have not generated that report already, you may do so using the following commands, which add a `tz` virtual device and use it to create clock-zoned.json starting from clock.json.

```
$ openaps vendor add openapscontrib.timezones
$ openaps device add tz timezones
$ git add tz.ini
$ openaps report add monitor/clock-zoned.json JSON tz clock monitor/clock.json
```

To test this alias, you may first run your loop manually from command line, then execute `openaps status-upload`, examine the output, and check that the new status is visible on the OpenAPS pill box on your Nightscout page. To automate the status upload each time the loop is executed you can simply add `status-upload` to your main OpenAPS loop alias. The OpenAPS pill box will show when the last time your loop ran. If you hover over it, it will provide critical information that was used in the loop, which will help you understand what the loop is currently doing.

The OpenAPS pill box has four states, based on what happened in the last 15 minutes: Enacted, Looping, Waiting, and Warning:

- Waiting is when OpenAPS is uploading, but hasn't seen the pump in a while
- Warning is when there hasn't been a status upload in the last 15 minutes
- Enacted means OpenAPS has recently enacted the pump
- Looping means OpenAPS is running but has not enacted the pump

Some things to be aware of:

- Make sure that the timezones for the pi (if need be you can use `sudo raspi-config` to change timezones), in your monitor/clock-zoned.json report, and the Nightscout website are all in the same time zone.
- The basal changes won't appear in Nightscout until the second time the loop runs and the corresponding upload is made.
- You can scroll back in time and at each glucose data point you can see what the critical information was at that time

Note: Remember to add `careportal` to Nightscout's `ENABLE` environment variable in case it is not already there.

#### Set up `ns` device

To get your OpenAps viewed onto your Nightscout site, start by using the following tool:

```
nightscout autoconfigure-device-crud
```

To view your data on your Nightscout site, start by doing the following: `nightscout autoconfigure-device-crud https://yourname.com yourplainapisecret`

So this would be your actual `https://myname.azurewebsites.net` or `https://myname.herokuapp.com`. Your `API_SECRET` is listed in your Azure or Heroku settings. To test this: `openaps use ns shell preflight` To get aliases:

```
curl -sg https://gist.githubusercontent.com/bewest/d3db9ca1c144b845382c885138a8f66e/raw/181c5d6f29cd
```

#### Sending glucose data:

```
openaps use ns shell format-recent-type tz entries monitor/glucose.json  | json -a dateString | wc -l
# Add it as a report
openaps report add nightscout/recent-missing-entries.json JSON ns shell format-recent-type tz entries
# fetch data for first time
openaps report invoke nightscout/recent-missing-entries.json

# add report for uploading to NS
openaps report add nightscout/uploaded-entries.json JSON  ns shell upload entries.json nightscout/rec
# upload for fist time.
openaps report invoke nightscout/uploaded-entries.json
```

#### Uploading Latest Treatments to Nightscout

In addition to uloading OpenAPS status, it also very beneficial to upload the treatment information from the pump into Nightscout. This removes the burden of entering this information into Nightscout manually.

Note that a `pumphistory-zoned.json` report is required, which can be generated from pumphistory.json using `tz`, following the approach described above for clock-zoned.json, including making sure to add it to your monitor-pump alias. In addition, if you haven't already created a requisite reports you should create that report and invoke it since it is required. Upon successful upload, the recent treatments will show up automatically on the Nightscount page.

Note: Currently extended boluses are not handled well and depending on the timing of the upload are either missed entirely or have incorrect information.

To upload treatments data to Nightscout, prepare you zoned glucose, and pump model reports, and use the following two reports:

```
openaps report add nightscout/recent-treatments.json JSON ns shell  format-recent-history-treatments
openaps report add nightscout/uploaded.json JSON  ns shell upload-non-empty-treatments  nightscout/re
```

Here are the equivalent uses:

```
openaps use ns shell format-recent-history-treatments monitor/pump-history.json model.json
openaps use ns shell upload-non-empty-treatments nightscout/recent-treatments.json
```

The first report runs the format-recent-history-treatments use, which fetches data from Nightscout and determines which of the latest deltas from openaps need to be sent. The second one uses the upload-non-empty-treatments use to upload treatments to Nightscout, if there is any data to upload.

To pull data: `openaps gather-clean-data`

To set up Nightscout reports: curl -sg https://gist.githubusercontent.com/bewest/d3db9ca1c144b845382c885138a8f66e/raw/522155bae1 reports.json | openaps import

To see your progress: `openaps do-everything`

Then again, to check your progress: `openaps do-everything` At this point, you should see treatment circles, information about the battery, etc.

To verify what was uploaded to Nightscout: `cat nightscout/uploaded.json`

#### Sending openaps status to Nightscout

```
openaps use ns shell status monitor/clock.json oref0-monitor/iob.json oref0-predict/oref0.json oref0-
```

You should see a lot of info. (Side note: the word "received" is spelled wrong.)

Make sure to save this as a report:

```
openaps report add nightscout/openaps-status.json JSON ns-status shell monitor/clock.json oref0-monit
```

Now it needs to be invoked to test that it is getting data. `openaps report invoke nightscout/openaps-status.json`

Test uploading it: `openaps use ns shell upload devicestatus.json nightscout/openaps-status.json`

If it works, save it as a report:

```
openaps report add nightscout/uploaded-recent-devicestatus.json JSON ns shell upload devicestatus.jso
```

#### Updating Aliases

Now those aliases we did earlier need adjustment for all of the recent work we just did:

```
openaps alias add report-nightscout "report invoke  nightscout/recent-treatments.json nightscout/uplo
```

SUCCESS!!

To upload to Nightscout, use: `openaps do-everything` To just test uploading to Nightscout, use: `openaps report-nightscout`

Make sure to backup all the work you have just done: `oref0 export-loop | tee backup-loop.json`

## 3.3 Phase 2: Predict

### 3.3.1 Phase 2: Creating an Open Loop

Phase 2 focuses on deploying tools to utilize a suitable algorithm to recommend necessary changes to basal rates. This is essentially an open-loop system, with you completing the loop by manually calculating what you would do in that scenario. This can be performed in real time or by using historical data and making retroactive suggestions. Pay special attention to situations where CGM readings are not smooth (after calibration, with a new sensor, or with errors such as ???) or when there are issues with data connectivity or fidelity. Assume there will be issues with connectivity.

Note: oref0—short for "OpenAPS Reference Design 0"—is our first (zero-th) implementation of the OpenAPS Reference Design. It consists of a number of "Lego block" tools that, when combined with the core openaps toolset, create a full closed loop artificial pancreas system—an OpenAPS implementation.

By this stage, you should have already set up your pump and cgm as openaps devices. You will now add the oref0 tools as virtual devices, create openaps reports for commonly used queries and calculations, and add openaps aliases that bring together those reports into higher-level activities. Finally, you can combine those into a single command (or small set of them) that can do everything required to collect data, make a treatment recommendation, and enact it on the pump. You'll also build "preflight" and safety checks into the loop.

### 3.3.2 Building preflight and other safety checks

Before moving on to consolidating all of these capabilities into a single alias, it is a good idea to add some error checking.

- There are several potential issues that may adveresely affect operation of the system. For example, RF communication with the pump may be compromised. It has also been observed that the CareLink USB stick may become unresponsive or "dead", requiring a reset of the USB ports. Furthermore, in general, the system should not act on stale data. Let's look at some approaches you may consider to address these issues.

Ensuring that your openaps implementation can't act on stale data could be done by deleting all of the report files in the `monitor` directory before the reports are refreshed. You may simply use `rm -f` bash command, which removes file(s), while ignoring cases when the file(s) do not exist. If a refresh fails, the data required for subsequent commands will be missing, and they will fail to run. For example, here is an alias that runs the required bash commands:

```
openaps alias add gather '! bash -c "rm -f monitor/*; openaps gather-profile && openaps monitor-cgm &
```

This example also shows how an alias can be constructed using bash commands. First, all files in `monitor` directory are deleted. Then, aliases are executed to generate the required reports. A similar approach can be used to remove any old `suggested.json` output before generating a new one, and to check and make sure oref0 is recommending a temp basal before trying to set one on the pump. You may want to make sure that your `enact` alias includes these provisions.

It's also worthwhile to do a "preflight" check that verifies a pump is in communication range and that the pump stick is functional before trying anything else. The oref0 `mm-stick` command can be used to check the status of the MM CareLink stick. In particular, `mm-stick warmup` scans the USB port and exits with a zero code on success, and non-zero otherwise. Therefore,

```
$ mm-stick warmup || echo FAIL
```

will output "FAIL" if the stick is unresponsive or disconnected. You may simply disconnect the stick and give this a try. If the stick is connected but dead, `oref0-reset-usb` command can be used to reset the USB ports

```
$ sudo oref0-reset-usb
```

Beware, this command power cycles all USB ports, so you will temporarily loose connection to a WiFi stick and any other connected USB device.

Checking for RF connectivity with the pump can be performed by attempting a simple pump command or report and by examining the output. For example,

```
$ openaps report invoke monitor/clock.json
```

returns the current pump time stamp, such as "2016-01-09T10:47:56", if the system is able to communicate with the pump, or errors otherwise. Removing previously generated clock.json and checking for presence of "T" in the newly created clock.json could be used to verify connectivity with the pump.

Collecting all the error checking, a `preflight` alias could be defined as follows:

```
$ openaps alias add preflight '! bash -c "rm -f monitor/clock.json && openaps report invoke monitor/c
```

In this `preflight` example, a wait period of 120 seconds is added using `sleep` bash command if the USB ports have been reset in an attempt to revive the MM CareLink stick. This `preflight` example also shows how bash commands can be chained together with the bash && ("and") or || ("or") operators to execute different subsequent commands depending on the output code of a previous command (interpreted as "true" or "false").

You may experiment using `$ openaps preflight` under different conditions, e.g. with the CareLink stick connected or not, or with the pump close enough or too far away from the stick.

At this point you are in position to put all the required reports and actions into a single alias.

#Using oref0 Tools

### 3.3.3 Add the oref0 Virtual Devices

In Phase 1, you added two physical medical devices to openaps—your pump and your cgm. This was done using the command `$ openaps device add` and then specifying the device name, type, and parameters. OpenAPS tools to gather system profile parameters such as pump settings, calculate the current insulin on board (IOB), and determine if the pump temp basal should be updated or not, are contained in the OpenAPS reference system oref0. Since there is no physical oref0 device, you are essentially adding it to the openaps environment as a virtual device or plugin.

First, you can add a catch-all oref0 device using

```
$ openaps device add oref0 process oref0
```

and then you can be more specific and add individual oref0 processes as virtual devices using the following commands:

```
$ openaps device add get-profile process --require "settings bg_targets insulin_sensitivities basal_p
$ openaps device add calculate-iob process --require "pumphistory profile clock" oref0 calculate-iob
$ openaps device add determine-basal process --require "iob temp_basal glucose profile" oref0 determi
```

In these commands, `--require` specifies the arguments required by each of the oref0 processes. Most of the arguments to the oref0 processes should look familiar to you from your experimentation with `openaps` tools earlier. Now it's time to put together reports that the oref0 processes use as inputs, as well as reports and aliases that invoke the oref0 processes themselves.

### 3.3.4 Organizing the reports

It is convenient to group your reports into `settings`, `monitor`, and `enact` directories. The `settings` directory holds reports you may not need to refresh as frequently as those in `monitor` (e.g. BG targets and basal profile, vs. pump history and calculated IOB). Finally, the `enact` directory can be used to store recommendations ready to be reviewed or enacted (sent to the pump). The rest of this section assumes that you have created `settings`, `monitor`, and `enact` as subdirectories in your openaps directory. The following shell command creates these three directories:

```
$ mkdir -p settings monitor enact
```

### 3.3.5 The get-profile process

The purpose of the `get-profile` process is to consolidate information from multiple settings reports into a single JSON file. This makes it easier to pass the relevant settings information to oref0 tools in subsequent steps. Let's look at what kind of reports you may want to set up for each of the `get-profile` process arguments:

- `settings` outputs a JSON file containing the pump settings:

```
    $ openaps report add settings/settings.json JSON pump read_settings
```

- `bg_targets` outputs a JSON file with bg targets collected from the pump:

```
    $ openaps report add settings/bg_targets_raw.json JSON pump read_bg_targets
```

If your pump OR CGM is European and displays mmol/L as opposed to mg/dl you will need to convert this "raw" file.

First install the unit conversion device to ensure all units will match.

```
 $ openaps device add units units
```

Go through the standard process of use, report add, report invoke for the 2 reports below.

For Blood Sugar Conversion The `units` function ensures that units will match. To use it, add units to your list of devices with:

```
$ openaps device add units units
```

To convert this "raw" file, we need to add a report that will perform `$ openaps use units bg_targets settings/bg_targets_raw.json` and output not to the screen but into a file called settings/bg_targets.json:

```
$ openaps report add settings/bg_targets.json JSON units bg_targets settings/bg_targets_raw.json
```

For Insulin Sensitivity

```
$ openaps report add settings/insulin_sensitivities.json JSON units insulin_sensitivities settings/in
```

- `insulin_sensitivities` outputs a JSON file with insulin sensitivites obtained from the pump:

```
    $ openaps report add settings/insulin_sensitivities_raw.json JSON <my_pump_name> read_insulin_se
```

- `basal_profile` outputs a JSON file with the basal rates stored on the pump in your basal profile

```
    $ openaps report add settings/basal_profile.json JSON <my_pump_name> read_basal_profile_std
```

- `preferences` is an exception: in contrast to the other settings above, `preferences` is not the result of an openaps report. It's a JSON file that should contain a single line with your maximum IOB, such as: `{"max_iob": 2}`. You can create this file by hand, or use the oref0-mint-max-iob tool to generate the file. The `max_iob` variable represents an upper limit to how much insulin on board oref0 is allowed to contribute by enacting temp basals over a period of time. In the example above, `max_iob` equals 2 units of insulin.

Make sure you test invoking each of these reports as you set them up, and review the corresponding JSON files using `cat`. Once you have a report for each argument required by `get-profile`, you can add a `profile` report:

```
$ openaps report add settings/profile.json text get-profile shell settings/settings.json settings/bg_
```

If you need to add a `profile` report that has the unit conversion, you can use this:

```
$ openaps report add settings/profile.json text get-profile shell settings/settings.json settings/bg_
```

Note how the `profile` report uses `get-profile` virtual device, with all the required inputs provided. At this point, it's natural to add an alias that generates all the reports required for `get-profile`, and then invokes the `profile` report that calls `get-profile` on them:

```
$ openaps alias add gather-profile "report invoke settings/settings.json settings/bg_targets_raw.jso
```

Remember, what you name things is not important - but remembering WHAT you name each thing and using it consistently throughout is key to saving you a lot of debugging time. Also, note that the name of your report and the name of the corresponding file created by the report are the same. For example, you invoke a report called "settings/settings.json" and the results are stored in "settings/settings.json". The corresponding output file is created by invoking the report.

### 3.3.6 The calculate-iob process

This process uses pump history and the result of `get-profile` to calculate IOB. The IOB is calculated based on normal boluses and basal rates over past several hours. At this time, extended boluses are not taken into account. The `get-profile` arguments, and suggested reports are as follows:

- `profile`: report for `get-profile`, as discussed above

- `pumphistory` stores pump history in a JSON file

```
$ openaps report add monitor/pumphistory.json JSON pump iter_pump_hours 4
```

In this example, pump history is over a period of 4 hours. Normally, you would want oref0 to operate based on pump history over the number of hours at least equal to what you assume is your active insulin time.

- `clock` outputs the current time stamp from the pump

```
$ openaps report add monitor/clock.json JSON pump read_clock
```

You can now add a report for the `calculate-iob` process:

```
$ openaps report add monitor/iob.json JSON calculate-iob shell monitor/pumphistory.json settings/pro
```

As always, it is a good idea to carefully test and examine the generated reports.

### 3.3.7 The determine-basal process

This process uses the IOB computed by `calculate-iob`, the current temp basal state, CGM history, and the profile to determine what temp basal to recommend (if any). Its arguments and reports could be setup as follow:

- `iob`: your report for `calculate-iob`
- `profile`: your report for `get-profile`
- `temp_basal` reads from pump and outputs the current temp basal state:

```
$ openaps report add monitor/temp_basal.json JSON pump read_temp_basal
```

- `glucose` reads several most recent BG values from CGM and stores them in glucose.json file:

```
$ openaps report add monitor/glucose.json JSON cgm iter_glucose 5
```

In this example, glucose.json will contain 5 most recent bg values.

Finally, a report for `determine-basal` may look like this:

```
$ openaps report add enact/suggested.json text determine-basal shell monitor/iob.json monitor/temp_ba
```

The report output is in suggested.json file, which includes a recommendation to be enacted by sending, if necessary, a new temp basal to the pump, as well as a reason for the recommendation.

If you are using a Minimed CGM (enlite sensors with glucose values read by your pump), you might get this error message when running this report `Could not determine last BG time`. That is because times are reported differently than from the Dexcom receiver and need to be converted first. See the section at the bottom of this page.

### 3.3.8 Adding aliases

You may want to add a `monitor-pump` alias to group all the pump-related reports, which should generally be obtained before running `calculate-iob` and `determine-basal` processes:

```
$ openaps alias add monitor-pump "report invoke monitor/clock.json monitor/temp_basal.json monitor/pu
```

For consistency, you may also want to add a `monitor-cgm` alias. Even though it's invoking only a single report, keeping this consistent with the `monitor-pump` alias makes the system easier to put together and reason about.

```
$ openaps alias add monitor-cgm "report invoke monitor/glucose.json"
```

### 3.3.9 Checking your reports

At this point you can call

```
$ openaps report show
```

and

```
$ openaps alias show
```

to list all the reports and aliases you've set up so far. You'll want to ensure that you've set up a report for every argument for every oref0 process and, *more importantly*, that you understand what each report and process does. This is an excellent opportunity to make some `openaps report invoke` calls and to `cat` the report files, in order to gain better familiarity with system inputs and outputs.

You can also test the full sequence of aliases and the that which depend on them:

```
$ rm -f settings/* monitor/* enact/*
$ openaps gather-profile
$ openaps monitor-pump
$ openaps monitor-cgm
$ openaps report invoke monitor/iob.json
$ openaps report invoke enact/suggested.json
```

It is particularly important to examine suggested.json, which is the output of the `determine-basal` process, i.e. the output of the calculations performed to determine what temp basal rate, if any, should be enacted. Let's take a look at some suggested.json examples:

```
{"temp": "absolute","bg": 89,"tick": -7,"eventualBG": -56,"snoozeBG": 76,"reason": "Eventual BG -56<1
```

In this example, the current temporary basal rate type is "absolute", which should always be the case. The current BG values is 89, which dropped from 96 by a "tick" value of -7. "eventualBG" and "snoozeBG" are oref0 variables projecting ultimate bg values based on the current IOB with or without meal bolus contributions, an average change in BG over the most recent CGM data in glucose.json, and your insulin sensitivity. The "reason" indicates why the recommendation is made. In the example shown, "eventualBG" is less than the target BG (100), no temp rate is currently set, and the temp rate required to bring the eventual BG to target is -0.435U/hr. Unfortunately, we do not have glucagon available, and the pump is unable to implement a negative temp basal rate. The system recommends the best it can: set the "rate" to 0 for a "duration" of 30 minutes. In the oref0 algorithm, a new temp basal rate duration is always set to 30 minutes. Let's take a look at another example of `suggested.json`:

```
{"temp": "absolute","bg": 91,"tick": "+6","eventualBG": -2,"snoozeBG": 65,"reason": "Eventual BG -2<1
```

In this case, the evenatual BG is again less than the target, but BG is increasing (e.g. due to a recent meal). The actual "tick", which is also referred to as "Delta", is larger than the change that would be expected based on the current IOB and the insulin sensitivity. The system therefore recommends canceling the temp basal rate, which is in general done by setting "duration" to 0. Finally, consider this example:

```
{"temp": "absolute","bg": 95,"tick": "+4","eventualBG": 13,"snoozeBG": 67,"reason": "Eventual BG 13<9
```

which is similar to the previous example except that in this case there is no temp basal rate to cancel. To gain better understanding of oref0 operation, you may want to also read Understanding oref0-determine-basal recommendations and spend some time generating and looking through suggested.json and other reports.

### 3.3.10 Enacting the suggested action

Based on suggested.json, which is the output of the `determine-basal` oref0 process, the next step is to enact the suggested action, i.e. to send a new temp rate to the pump, to cancel the current temp rate, or do nothing. The

approach one may follow is to setup an `enacted.json` report, and a corresponding `enact` alias. Thinking about how to setup the `enact` report and alias, you may consider the following questions:

- Which pump command could be used to enact a new basal temp, if necessary, and what inputs should that command take? Where should these inputs come from?
- How could a decision be made whether a new basal temp should be sent to the pump or not? What should `enact` do in the cases when no new temp basal is suggested?

Once you setup your `enact` alias, you should plan to experiment by running the required sequence of reports and by executing the `enact` alias using `$ openaps enact`. Plan to test and correct your setup until you are ceratin that `enact` works correctly in different situations, including recommendations to update the temp basal, cancel the temp basal, or do nothing.

In order to ensure that your pump is able to accept the temp basal suggestion, ensure that the temp basal setting, on the pump itself is set to "Insulin Rate (U/H)". This can be found in Act>basal>Temp basal type.

### 3.3.11 Cleaning CGM data from Minimed CGM systems

If you are using the Minimed Enlite system, then your report for `iter_glucose` uses your pump device because the pump is the source of your CGM data. Unfortunately, the pump reports CGM data a bit differently and so your glucose.json file needs cleaning to align it with Dexcom CGM data. The simplest way to handle this is with this excellent plug-in:

https://github.com/loudnate/openaps-glucosetools

After installing glucosetools, add the vendor and device with:

```
$ openaps vendor add openapscontrib.glucosetools
$ openaps device add glucose glucosetools
```

Now you can create a report to clean your glucose data like this:

```
openaps report add monitor/glucoseclean.json JSON glucose clean monitor/glucose.json
```

And you should then make sure that your enact/suggested.json report uses monitor/glucoseclean.json instead of monitor/glucose.json. You can add the `clean` report to your `monitor-cgm` alias, as long as it comes after the `iter_glucose` report.

Note that if you use Nightscout visualization as described later, you can use the built-in tool `mm-format-ns-glucose` to help formatting the Minimed glucose data. If you do, run the tool against the original `iter-glucose` output (monitor/glucose.json), *not* the output from glucosetools.

### 3.3.12 Running an open loop with oref0

To pull all of oref0 together, you could create a "loop" alias that looks something like `openaps alias add loop '! bash -c "openaps monitor-cgm 2>/dev/null && ( openaps preflight && openaps gather && openaps enact) || echo No CGM data."'`. If you want to also add some retry logic to try again if something failed, you could then do something like `openaps alias add retry-loop '! bash -c "openaps preflight && until( ! mm-stick warmup || openaps loop); do sleep 5; done"'`.

Once all that is working and tested, you will have a command that can be run manually or on a schedule to collect data from the pump and cgm, calculate IOB and a temp basal suggestion, and then enact that on the pump.

## 3.4 Phase 3: Control

### 3.4.1 Phase 3: Understanding Your Open Loop

Phase 3 entails understanding what the open loop is recommmending, and why. You should run with an "open loop" for enough days to decde what your max basal setting should be, and also observe how often you disagree or decide to counteract what the loop is recommending.

### 3.4.2 Understanding the output of oref0-determine-basal

The key logic behind any oref0 implementation of OpenAPS lies in the oref0-determine-basal.js code, which is what takes all of the inputs you've collected and makes a temp basal recommendation you can then enact if appropriate. As such, it is important to understand how determine-basal makes its decisions, and how to interpret its output, so you can decide for yourself whether the recommendations it is making are appropriate for your situation, or if further adjustments are required before closing the loop or letting it run unattended.

The recommendation is to run for several days in "open loop" mode, watching the output, in order to decide what your "max basal" setting should be. Based on how often you disagreed or counteracted what the loop was recommending, this might influence how you set your max basal.

#### Summary of inputs

The determine-basal algorithm requires a number of inputs, which are passed in JSON files such as iob.json, currenttemp.json, glucose.json, profile.json, and optionally meal.json. When running oref0-determine-basal.js with the appropriate inputs, the first thing you'll see is a summary of all the provided inputs, which might look something like this:

```
{"carbs":0,"boluses":0}
{"delta":-2,"glucose":110,"avgdelta":-2.5}
{"duration":0,"rate":0,"temp":"absolute"}
{"iob":0,"activity":0,"bolussnooze":0,"basaliob":0}
{"carbs_hr":28,"max_iob":1,"dia":3,"type":"current","current_basal":1.1,"max_daily_basal":1.3,"max_ba
```

- The first line is meal.json, which, if provided, allows determine-basal to decide when it is appropriate to enable Meal Assist.
- The second line is from glucose.json, and represents the most recent BG, the change from the previous BG (usually 5 minutes earlier), and the average change since 3 data points earlier (usually 15 minutes earlier).
- The third line is the currently running temporary basal. A duration of 0 indicates none is running.
- Fourth is the IOB and insulin activity summary. Insulin activity is used (when multiplied by ISF) to calculate BGI, which represents how much BG should be rising or falling every 5 minutes based solely on insulin activity. Basal IOB excludes the IOB effect of boluses, and Bolus Snooze is used in determining how long to avoid low-temping after a bolus while waiting for any carbs to kick in.
- Fifth is the contents of profile.json, which contains all of the user's relevant pump settings, as well as their configured maximum (basal) IOB.

#### Output

After displaying the summary of all input data, oref0-determine-basal outputs a recommended temp basal JSON, which includes an explanation of why it's recommending that. It might look something like this:

```
{"temp":"absolute","bg":110,"tick":-2,"eventualBG":95,"snoozeBG":95,"mealAssist":"Off: Carbs: 0 Bolus
```

In this case, BG is 110, and falling slowly. With zero IOB, you would expect BG to be flat, so the falling BG generates a "deviation" from what's expected. In this case, because avgdelta is -2.5 mg/dL, vs. BGI of 0, that avgdelta is extrapolated out for the next 30 minutes, resulting in a deviation of -30 mg/dL. That is then applied to the current BG to get an eventualBG of 80. There is no bolussnooze IOB, so snoozeBG is also 80, and because (among other things) avgdelta is negative, mealAssist remains off. To correct from 80 up to 115 would require a -2.65U/hr temp for 30m, and since that is impossibly low, determine-basal recommends setting a temp basal to zero and stopping all insulin delivery for now.

#### Exploring further

For each different situation, the determine-basal output will be slightly different, but it should always provide a reasonable recommendation and list any temp basal that would be needed to start bringing BG back to target. If you are unclear on why it is making a particular recommendation, you can explore further by searching lib/determine-basal/determine-basal.js (the library with the core decision tree logic) for the keywords in the reason field (for example, "setting" in this case would find a line (`rT.reason += ", setting " + rate + "U/hr";`) matching the output above, and from there you could read up and see what `if` clauses resulted in making that decision. In this case, it was because (working backwards) `if (snoozeBG > profile.min_bg)` was false (so we took the `else`), but `if (eventualBG < profile.min_bg)` was true (with the explanatory comment to tell you that means "if eventual BG is below target").

If after reading through the code you are still unclear as to why determine-basal made a given decision (or think it may be the wrong decision for the situation), please join the #intend-to-bolus channel on Gitter, paste your output and any other context, and we'll be happy to discuss with you what it was doing and why, and whether that's the best thing to do in that and similar situations.

## 3.5 Phase 4: Iterate and Improve

### 3.5.1 Phase 4: Starting to Close the Loop

Phase 4 focuses on creating a schedule to automate the manual system you developed. Again, at this stage testing is critical and output of the system should be tracked and validated over a series of time, and include thorough edge case testing as you move from communicating with the pump and CGM to closing the loop. Don't test overnight until you are supremely confident in the operation of your system - remembering that you hold responsibility for the setup and operation of your system. You may want to have someone nearby the first time you run the system overnight, in addition to setting alarms to test and watch the system.

### 3.5.2 Using cron to create a schedule for your loop

You should use cron to create a schedule for your loop. Use `oref0 cron-5-minute-helper` to generate a simple cron job. It can be imported into crontab using `oref0 cron-5-minute-helper do-loop | crontab -`. By default, it will list a suggested cron job that runs once every 5 minutes.

Here's an example:

```
$ oref0 cron-5-minute-helper openaps do-foo-bar
SHELL=/bin/bash
PATH=/home/bewest/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/lo

*/5 * * * * (cd /home/bewest/src/openaps/docs && time openaps do-foo-bar) 2>&1 | logger -t openaps-lo
```

It prepares a cron template to change to the current directory and runs whatever was specified, sending all output to syslog.

### 3.5.3 Observing the closed loop

After you have created a schedule with cron, your loop will automatically run.

#### Starting with low glucose suspend mode

You may notice that the loop is only going to temp you til your netIOB (this means the net amount of insulin compared to your normal basal rates; so if you temp to 0 and your basal is usually 1.3; you will end up with negative netIOB) returns to 0. This is an important safety feature to help you observe the loop for three days in this mode.

At this stage, if you consistently see net negative insulin amounts at the same time every day (use Nightscout reports or similar to observe), then you may be over-bolusing for meals and/or your basal rates are too high the 1.5-2 hours before the low BGs are occurring.

If you observe this, you'll need to tweak your basals or consider how you bolus for meals - your I:C ratio may be off.

Make sure you check this out before moving to the next stage of tuning the loop.

### 3.5.4 Troubleshooting the closed loop

If you haven't read this enough already: the DIY part of this is really important. If you've been copying and pasting, and not understanding what you're doing up to this point - please stop. That's dangerous. You should be testing and validating your work, and asking questions as you go if anything is unclear. (And, if this documentation annoys you enough, put in a PR as you go through each part to update/improve the documentation to help the next person! We've all been there.) :)

That being said, at this stage, you have both a manual loop and a schedule (using cron) to create an automated loop. At this point, you're in the "test and watch" phase. In particular, you should make sure the loop is recommending and enacting the types of temporary basal rates that you might do manually; and that the communication is working between the devices.

Additionally, most loopers, after automating their system with cron jobs, begin to think through some of the following things and run the following tests, including unit tests:

#### How often should the cron run?

Think about how often you get new BG data and may want to act on it. Or, time how long it takes your loop to run, and add another minute to that. You probably want to add preflight checks to make sure a loop is not already running before your next one starts.

#### What should BG target range be?

In the early testing, the OpenAPS settings may cause your BG to go both high and low.

It's tempting to set your targets to "perfect" on day one, and start your looping with those values. The problem with this is that if the algorithm incorrectly gives you too much insulin you don't have very much room to handle emergencies.

To start off, you should set your glucose target range "high and wide". Once you can reproducibly get your sugars in a wider and higher band without going low, you can then *slowly* reduce the target range to your ideal range.

You should work toward a long-term goal here, rather than trying to do everything on day one.

Additionally, as you think about the lower end of your target range, remember the timing of your insulin activity and the fact that negative insulin corrections take about the same amount of time to go into effect; thus, you wouldn't want your low end of the target range set below 90, for example - otherwise the system will not be able to prevent lows by reducing the insulin.

### What should pump settings be?

You should set your maximum temporary basal limit on your pump to a reasonable value, to try and make sure that you don't go low by accident.

To start off, you should think about taking the largest basal rate in your profile and multiply it by a 1.3. Set that as your maximum temporary basal on the pump.

Once you're happy things are functioning correctly, you can increase this value to about 2x your basal.

Note that for children especially this can vary a lot based on age, weight, and activity. Err on the side of caution.

### What happens if the system gets out of range or gets bad data?

Test the range of the system. What happens if you walk out of range of the Carelink stick? What happens to the temporary basal rate? What happens with your cron job? What do you need to be aware of? Apply the same set of questions and thinking for other scenarios, including if you go out of range of your CGM and/or get ??? or another CGM error message.

Make sure you understand the limits of the transmitter and these other errors, especially in an overnight situation , and what the system can and can't do when you are out of effective range.

As your tests extends from minutes to hours, you'll want to check at least these scenarios: data corruption, lack of data, lack of connectivity, and other non-ideal operating conditions.

### Unit testing

Additionally, you may want to consider some unit testing. There is a basic unit testing framework in oref0 that you can use, and add to.

#### To help with unit test cases:

If you'd like to help out with defining all the desired behaviors in the form of unit test cases:

1. Please clone / checkout [oref0] (https://github.com/openaps/oref0)
2. Type `sudo npm install -g mocha` and `sudo npm install -g should`
3. You should then be able to run `make` from the ~/src/oref0 directory to run all of the existing unit tests, or something like `mocha -c tests/determine-basal.test.js 2>&1 | less -r` to run just the determine-basal tests.
4. As you add additional unit tests, you'll want to run `make` again after each one.

#### How to add more test cases:

We'll want to cover every section of the code, so if you see a "rT.reason" in bin/oref0-determine-basal.js that doesn't have a corresponding "output.reason.should.match" line in an appropriate test in tests/determine-basal.test.js, then you should figure out what glucose, temp basal, IOB, and profile inputs would get you into that section of the code (preferably representing what you're likely to see in a real-world situation), and create a test case to capture those

inputs and the desired outputs. Then run the tests and see if your test passes, and the output looks reasonable. If not, then modify your test case accordingly, or if you think you've found a bug in determine-basal.js, ask on Gitter.

## 3.6 Phase 5: Tuning the Closed Loop

### 3.6.1 Phase 5: Tuning the Closed Loop

At this point, you have a closed loop that can run overnight, but it's been in 'low glucose suspend mode'. At this stage, you'll advance to changing your settings and tuning your targets in incremental steps to find what the right settings are for you.

### 3.6.2 Going beyond low glucose suspend mode

You may have noticed that in the previous phase (4), in observing low glucose suspend mode, the loop did not temp you to get your netIOB above 0.

Once you have spent several days observing the loop in the previous mode and made sure your basals and bolus strategies are in good shape, you may consider moving to the next step.

This means adjusting your max iob amount.

Keep in mind this is one of the key safety features of OpenAPS. You do NOT want this to be a super large amount. The point of this setting is to ensure that the loop can not excessively high temp you; if you need high temps consistently to get you to this amount, your baseline basals are off OR you missed a meal bolus OR you are sick OR there is some other extenuating circumstance; but in all of these cases, they should require manual intervention and you should not expect the loop to solve for this.

A good rule of thumb is for max iob to be 3 times your highest basal rate.

(This means it should be approximate to your other settings; not an absolute amount that you set without thinking about it.)

### 3.6.3 Tuning your targets

After you adjust your max iob and go beyond low glucose suspend mode, run the system overnight under close observation with the following considerations around targets:

- You should start with high targets and a good safety margin. For example, you might start with your target at 150 to see how the system does. OpenAPS has a "min" target floor which prevents you from setting it below 90.
- Before adjusting your target, you should have at least one night with zero low alarms (in three days) before considering dropping the max target below 160.
- Each time you adjust your target, it should be no more than 5-10 points at a time, again observing the outcomes over a few days.

If you are going low overnight for three+ days, your "min" target may need to be raised.

After you tune your targets, min can be set to be equivalent of max. However, when you first start, start with a 10-point range (i.e. 150-160).

## 3.7 Phase 6: Iterate and Improve the Closed Loop

### 3.7.1 Phase 6: Iterating on Your Closed Loop

Now that you're looping, congrats! Make sure you follow these steps - you're not done yet. You'll want to let us know that you are looping, and continue testing.

You in particular may want to test over a 24 hour period in the course of 3 days (doesn't have to be consecutive) to see what the system does when you do meal boluses with the system on, if you are interested in looping during the day.

Watching the system over long periods of time will help you continue to tweak how you bolus for meals and adjust baseline basals.

At this point, you can also investigate some of the in-development advanced features, like meal-assist and auto-sensitvitiy adjustment. You can also explore some of the hardware pieces in progress, as noted on the hardware page.

### 3.7.2 So you think you're looping? Now keep up to date!

If you've gone "live" with your loop, congratulations! You'll probably want to keep a very close eye on the system and validate the outputs for a while. (For every person, this amount of time varies).

One important final step, in addition to continuing to keep an eye on your system, is letting us know that you are looping.

**This is important in case there are any major changes to the system that we need to notify you about**. One example where this was necessary is when we switched from 2015 to 2016: the dates were incorrectly reporting as 2000, resulting in incorrect IOB calculations. As a result, we needed to notify current loopers so they could make the necessary update/upgrade.

#### After you have looped for three consecutive nights:

So that we can notify you if necessary, please fill out this form if you have been looping for 3+ days. Your information will not be shared in any way. You can indicate your preferred privacy levels in the form. As an alternative, if you do not want to input info, please email dana@openaps.org. Again, this is so you can be notified in the case of a major bug find/fix that needs to be deployed.

### 3.7.3 Testing during the day

You probably started looping for nighttime; but some people also choose to loop during the day. Here are some things to think about:

- Run the loop over 24 hours collectively in 3 days (doesn't have to be consecutive), and simply observe what the system does when you do meal boluses. You may want to go back to watching determine-basal to see understand what the loop is doing, and understanding `snoozeBG` and similar.
- Continue to observe the net basal insulin delivery after one week of looping. See what times of day the system is consistently leaving you with neg negative amounts of IOB to counteract or prevent low BGs; again, you may still be overbolusing and/or have basal rates set too high as a baseline.

### 3.7.4 Advanced features

Once you have several days or weeks of looping during the day and night, you may then want to consider some of the in-development additional features. See below for a high-level overview; and then see subsequent pages for details

about setting these up. These are NOT turned on by default, and require manual configuration and additional work to use them.

#### Meal-assist

If you choose to enable the optional meal-assist feature, then after you give yourself a meal bolus, the system can high-temp more quickly after a meal bolus IF you enter carbs reliably. So before considering meal-assist, you must be willing to enter carbs reliably, either through the pump's bolus wizard, or through the Nightscout Care Portal. If you don't want to do that, this feature won't be usable.

Like all features and steps, you'll want to carefully enable, test, and observe the outcomes of this feature.

#### Auto-sensitivity mode

Wouldn't it be great if the system knew when you were running sensitive or resistant? That's what we thought, so we created "auto-sensitivity mode". If you explicitly configure this additional feature, it will allow the system to analyze historical data on the go and make adjustments if it recognizes that you are reacting more sensitivite (or conversely, more resistant) to insulin than usual. It will then make micro adjustments to your basals.

#### Battery monitoring

Because running OpenAPS requires frequent communication with your pump, your pump battery tends to drain more quickly than you'd experience when not looping. Some users have had good experiences with Energizer Ultimate Lithium AAA batteries (getting ~1.5weeks) rather than alkaline batteries (getting ~2-3 days). Regardless of whether you use alkaline or lithium, you may want to consider a Nightscout alarm to alert you to when the battery is running low. You can do this by setting (in your Nightscout config vars) `PUMP_WARN_BATT_V` to 1.39 (if using lithium batteries, as is most common) and adding `battery` to your `PUMP_FIELDS` so that voltage is displayed on your Nightscout site.

The 1.39 voltage warning will give you many hours (reportedly ~8+) heads up that you will need to change your battery. If you don't change the battery in time and end up with a "low battery" warning on the pump, the pump will still function, but RF communications will be turned off and you will not be able to loop until you put a new battery in.

### 3.7.5 Configuring Automatic Sensitivity and Meal Assist Mode

For more information review https://github.com/openaps/oref0/issues/58

1. Install the latest dev branch of `oref0`:

```
sudo npm install -g git://github.com/openaps/oref0.git'#dev'
```

1. Next in order for the new auto-sensitivity report to run, you need to have at least 24 hours worth of pump history data and enough bg readings (24 hours). In your `openaps.ini` apply the following changes:

```
[report "monitor/glucose.json"]
device = cgm
use = iter_glucose
reporter = JSON
count = 288
```

(NOTE: if using Nightscout, add `count=288` to your `entries.json` API call as a querystring parameter)

One way to do this is to go to your openaps directory and edit the `ns-glucose.ini` file. Depending on how you've implemented it, it should look something like this (please note the `?count=288`):

```
[device "curl"]
fields =
cmd = bash
vendor = openaps.vendors.process
args = -c "curl -s https://[Your URL]/api/v1/entries.json?count=288 | json -e 'this.glucose = this.sg
```

If your `glucose.json` does not have enough entries you will see a warning when running your `auto-sens.json` report:

```
Error: not enough glucose data to calculate autosens.
```

1. After applying the above change you need to add a new `auto-sens` device and an `auto-sens` report. Run this command to create the `auto-sens` report:

```
openaps device add auto-sens process --require "glucose pumphistory
insulin_sensitivities basal_profile profile" oref0 detect-sensitivity
```

If the command exectuted properly, the contents of `openaps.ini` should contain:

```
[device "auto-sens"]
vendor = openaps.vendors.process
extra = auto-sens.ini
```

Another new file named `auto-sens.ini` should have been created, and it should contain:

```
[device "auto-sens"]
fields = glucose pumphistory insulin_sensitivities basal_profile profile
cmd = oref0
args = detect-sensitivity
```

1. In order for `auto-sens` to run properly, you need to make sure you pull enough history from your pump - 24 hours plus however many yours you have set for your DIA. To do this, you will create a new report called `pumphistory-24h.json`:

```
openaps report add settings/pumphistory-24h.json JSON pump iter_pump_hours 28
```

(NOTE: the `28` assumes a 4h DIA - please adjust accordingly if your DIA is longer.)

1. Once the device is created, we need to create the `auto-sens.json` report. Run this command to create the `auto-sens.json` report:

```
openaps report add settings/auto-sens.json text auto-sens shell monitor/glucose.json settings/pumphis
```

Now invoke the report to test:

```
openaps invoke report settings/auto-sens.json
```

1. Next we need to add the `auto-sens.json` report to the `oref0-determine-basal` device. In `openaps.ini` make sure your `oref0-determine-basal` looks similar to this:

```
[device "oref0-determine-basal"]
fields = iob current-temps glucose profile **auto-sens** meal
cmd = oref0-determine-basal
vendor = openaps.vendors.process
args =
```

(NOTE: in the `fields` above, `meal` should only be present if meal assist is configured)

1. At this point, in the process you should already have an `enact/suggested.json` report. Edit your `openaps.ini` file and add the bottom line to that report:

```
[report "enact/suggested.json"]
profile = settings/profile.json
use = shell
reporter = text
current-temps = monitor/temp-basal-status.json
device = oref0-determine-basal
iob = monitor/iob.json
glucose = monitor/glucose.json
meal = monitor/meal.json
auto-sens = settings/auto-sens.json
```

(NOTE: as stated above, if you do not have meal assist enabled, do not include the `meal` line)

1. Based on the configuration of the basic loop, it is recommended that the `settings/auto-sens.json` be added to the `gather-profile` alias:

```
gather-profile report invoke settings/settings.json settings/bg_targets.json settings/insulin_sensiti
```

and that the `gather` alias be adjusted to make sure `gather-profile` is at the end. This is because the `settings/auto-sens.json` report depends upon elements from the preceding two aliases to run.

```
gather !  bash -c "rm -f monitor/*; openaps monitor-cgm && openaps
monitor-pump && openaps gather-profile"
```

Note. Your loop should run without `auto-sens.json` report but if you don't pass that as an input you will see the following message while executing `oref0-determine-basal.js`:

```
Optional feature Auto Sensitivity not enabled:  { [Error: ENOENT: no such file or directory, open 'or
```

Here is an example of running the loop with Auto Sensitivity feature enabled:

```
reporting oref0-prepare/mm-normalized.json
.....................................................................................
p=0.51: -0.33, -0.33, 0.22
p=0.50: -0.33, -0.33, 0.13
p=0.49: -0.67, -0.36, -0.03
p=0.48: -0.67, -0.39, -0.11
p=0.47: -0.67, -0.46, -0.24
p=0.46: -0.67, -0.50, -0.38
p=0.45: -1.00, -0.54, -0.42
p=0.44: -1.24, -0.57, -0.47
p=0.43: -1.33, -0.58, -0.63
p=0.42: -1.49, -0.60, -0.67
p=0.41: -1.67, -0.62, -0.78
p=0.40: -1.67, -0.65, -0.88
p=0.39: -1.67, -0.66, -0.95
p=0.38: -2.00, -0.67, -1.12
p=0.37: -2.00, -0.68, -1.57
p=0.36: -2.00, -0.69, -1.67
p=0.35: -2.33, -0.69, -1.73
p=0.34: -2.33, -0.70, -1.89
p=0.33: -2.43, -0.74, -2.03
p=0.32: -2.67, -0.76, -2.16
p=0.31: -3.00, -0.77, -2.36
p=0.30: -3.00, -0.82, -2.55
Mean deviation: 0.76
Sensitivity within normal ranges
Basal adjustment 0.00U/hr
Ratio: 100%: new ISF: 42.0mg/dL/U
{"carbs":0,"boluses":0,"mealCOB":10}
```

```
{"ratio":1}
{"delta":-4,"glucose":129,"avgdelta":-6}
{"duration":17,"rate":1.125,"temp":"absolute"}
{"iob":0.089,"activity":0.0119,"bolussnooze":0,"basaliob":0.089,"netbasalinsulin":1.4,"hightempinsuli
{"max_iob":3,"type":"current","dia":3,"current_basal":1.15,"max_daily_basal":1.15,"max_basal":3,"min_
oref0-get-profile://text/shell/oref0-predict/profile.json
reporting oref0-predict/profile.json
oref0-calculate-iob://text/shell/oref0-predict/iob.json
reporting oref0-predict/iob.json
auto-sens://text/shell/oref0-monitor/auto-sens.json
reporting oref0-monitor/auto-sens.json
oref0-determine-basal://text/shell/oref0-predict/oref0.json
reporting oref0-predict/oref0.json
{
"temp": "absolute",
"bg": 129,
"tick": -4,
"eventualBG": 104,
"snoozeBG": 104,
"mealAssist": "Off: Carbs: 0 Boluses: 0 Target: 110 Deviation: -21 BGI: -2.5",
"reason": "Eventual BG 104>100 but Avg. Delta -6.00 < Exp. Delta -2.3, temp 1.125 ~ req 1.15U/hr"
}
No recommendation to send
```

CHAPTER 4

# Resources

## 4.1 Making your first PR (pull request)

At some point it will be suggested to you that you make a PR. PR is short for pull request. It's actually not too hard to do one and it is a great way to contribute. This documentation is here because people like you made PRs.

- The general idea is to make edits and improvements to code or document by making a copy of the repository you'd like to change.
- Double checking that they your edits look good to you on your copy.
- Make a few notes for what you did so people can understand why you made the change.
- Then do a pull request, which is to ask the administrators of the repository to pull your changes back into the appropriate branch of the main repository.
- They will do a quick review and merge your changes in, or comment if there are errors that need fixing first, or if it's a large enough change that it needs to go to another branch like dev for further work before being merged to master.

OK, let's get started. For our example we are going to make an edit to the openaps docs repository.

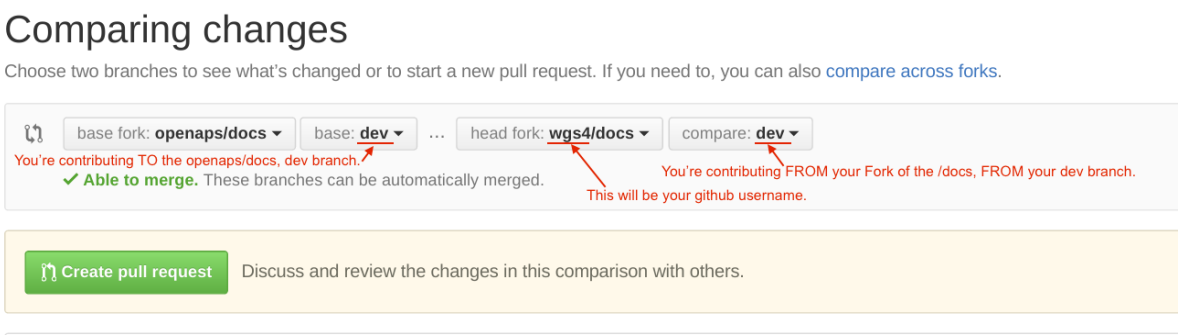
1. Go to https://github.com/openaps/docs and hit Fork in the upper right to make your own copy of the repository.
2. Github will automatically take you to your copy (notice in the address bar you are now in your own personal github directory)
3. Now we need to find the file we want to edit. Click through the directory structure until you find and are looking at the content of the file you want to change.
4. Next, press the pencil icon in the upper right next to the trash can icon.
5. Make edits to the file as necessary.
6. Next we want to commit our changes. But first we should note what we changed and why. Be sure to put a one liner explaining the why of making the changes you did.
7. Commit the changes.
8. Now look and make sure everyting you changed looks like you meant it to (no typos, etc). If any problems, go back and edit again and save again.

We now have an improved file that we want to be pulled back into the openaps/docs repository at https://github.com/openaps/docs

1. Go to https://github.com/[YOUR_GITHUB_USERNAME]/docs

- Or you can go to https://github.com and then click on "docs" in the "Your repositories" section in the lower right. Both methods will get you to the right place.

1. Click the green "New pull request" button
2. Under the Compare Changes heading, click "compare across forks"
3. Set up the the branches you are targeting. The easiest way of thinking about the branch range is this: the base branch is where you think changes should be applied, the head branch is what you would like to be applied.
4. So, choose the base fork as openaps/docs and then the base as master (or whichever branch you edited). The head fork is going to be youraccount/docs and the base as master (unless this is a large change that needs to go to dev first).




5. It should show the list of changes you made. If not, you did something wrong so stop here and ask for help. If the list looks like your changes then put a note in there to what the overarching reason for the changes are (in your case you only made one, but you could have made a bunch). Create the PR.

It will now be in a list of PR's that the team will review and potentially give feedback on before committing to the main documentation for openaps!

Congrats, you made your first contribution!

PS, your fork will still be sitting on your own personal github account. You can delete it if you are done with it. In the future, be sure to pull a fresh copy from github.com/openaps/docs before making new edits.

## 4.2 Technical Resources

These represent a small selection of guides, tutorials, and quick references for some of the tools used to develop and document OpenAPS.

### 4.2.1 Raspberry Pi

Raspberry Pi Documentation

Pi Filler, Pi Finder, and Pi Copier

### 4.2.2 Git and GitHub

Official Git Documentation

Atlassian (BitBucket) Git Tutorials

Code School Interactive Git Introduction

Codecademy Git Course

### 4.2.3 Linux Shell / Terminal

Learn UNIX in 10 Minutes

Codecademy Command Line Course

Cron How To Guide

### 4.2.4 Python

Official Python Documentation

Learn Python the Hard Way

Automate the Boring Stuff with Python

Codecademy Python Course

Python 2.7 Quick Reference

NumPy for MATLAB Users

### 4.2.5 Useful Apps

Fing (Android and Apple): Identify IP address of devices on a network. Useful for finding the IP address of RPi on new networks.

Hotspot Manager (Android): Itentify IP address of devices on a hotspot. Useful for finding the IP address of RPi on hotspots.

JuiceSSH (Android): SSH client for Android devices

### 4.2.6 Markdown syntax

Daring Fireball (John Gruber) Markdown Introduction

Macdown: Open-source Markdown editor for OS X

StackEdit: Online Markdown editor

## 4.3 Troubleshooting

Even those who follow this documentation precisely are bound to end up stuck at some point. This could be due to something unique to your system, a mistyped command, actions performed out of order, or even a typo in this guide. This section provides some tools to help diagnose the issue as well as some common errors that have been experienced and resolved before. If you get stuck, try re-reading the documentation again and after that, share what you've been working on, attempted steps to resolve, and other pertinent details in #intend-to-bolus in Gitter when asking for help troubleshooting.

### 4.3.1 Generally useful linux commands

More comprehensive command line references can be found here and here. For the below, since these are basic linux things, also try using a basic search engine (i.e. Google) to learn more about them and their intended use.

`$ ls -alt` (List all of the files in the current directory with additional details.)

`$ cd` (Change directory)

`$ pwd` (Show the present working directory (your current location within the filesystem).)

`$ sudo <command>`

`$ tail -f /var/log/syslog`

`$ df -h`

`$ ifconfig`

`$ cat <filename>` (Display the contents of the file.)

`$ nano <filename>` (Open and edit the file in the nano text editor.)

`$ stat <filename>`

`$ pip freeze`

`$ sudo reboot`

`$ sudo shutdown -h now` (The correct way to shut down the Raspberry Pi from the command line. Wait for the green light to stop blinking before removing the power supply.)

`$ dmesg` (Displays all the kernel output since boot. It's pretty difficult to read, but sometimes you see things in there about the wifi getting disconnected and so forth.)

`uptime`

[add something for decocare raw logging]

### 4.3.2 Dealing with the CareLink USB Stick

The `model` command is a quick way to verify whether you can communicate with the pump. Test this with `$ openaps use <my_pump_name> model`.

If you can't get a response, it may be a range issue. The range of the CareLink radio is not particularly good, and orientation matters; see range testing report for more information.

If you still can't get a response, trying unplugging and replugging the CareLink stick.

Once you're setting up your loop, you may also want to oref0-reset-usb (`oref0-reset-usb.sh`) if mm-stick warmup fails, to reset the USB connection. It can help in some cases of CareLink stick not responding. Just note that during USB reset you will loose your Wi-Fi connection as well.

### 4.3.3 Dealing with a corrupted git repository

OpenAPS uses git as the logging mechanism, so it commits report changes on each report invoke. Sometimes, due to "unexpected" power-offs (battery dying, unplugging, etc.),the git repository gets broken. When it happens you will receive exceptions when running any report from openaps. As git logging is a safety/security measure, there is no way of disabling these commits.

To fix a corrupted git repository you can run `oref0-fix-git-corruption.sh`, it will try to fix the repository, and in case when repository is definitly broken it copies the remainings in a safe place (`tmp`) and initializes a new git repo.

Warning: do not run any openaps commands with sudo in front of it `sudo openaps`. If you do, your .git permissions will get messed up. Sudo should only be used when a command needs root permissions, and openaps does not need that. Such permission problems can be corrected by running `sudo chown -R pi.pi .git` in the openaps directory. If you are using an Intel Edison, run `sudo chown -R edison.users .git`.

### 4.3.4 Environment variables

If you are getting your BG from Nightscout or you want to upload loop status/resuts to Nightscout, among other things you'll need to set 2 environment variabled: `NIGHTSCOUT_HOST` and `API_SECRET`. If you do not set and export these variables you will receive errors while running `openaps report invoke monitor/ns-glucose.json` and while executing `ns-upload.sh` script which is most probably part of your `upload-recent-treatments` alias.Make sure your `API_SECRET` is in hashed format. Please see this page for details. Additionally, your `NIGHTSCOUT_HOST` should be in a format like `http://yourname.herokuapp.com` (without trailing slash). For the complete visualization guide use this page from the OopenAPS documentation.

### 4.3.5 Common error messages

#### Don't have permission, permission not allowed, etc

The command you are running likely needs to be run with root permissions, try the same command again with `sudo` in front of it

#### json: error: input is not JSON

```
json: error: input is not JSON: Unexpected '<' at line 1, column 1:
        <head><title>Document Moved</title></head>
```

This error usually comes up when you have pulled a file down from Nightscount that was an invalid file. Typcially you might see this when trying to pull down treatments. Make sure that you have your HOST and API_KEY set correctly at the top of your cron, in your ~/.profile

#### Could not parse carbratio_date when invoking profile report

```
Could not parse carbratio_data.
Feature Meal Assist enabled but cannot find required carb_ratios.
```

This error may occur when you invoke `settings/profile.json` report.

Check report definition in `openaps.ini`. If you have line `remainder = []` change it to `remainder =`

Below is correct definition

```
[report "settings/profile.json"]
use = shell
bg_targets = settings/bg_targets.json
settings = settings/settings.json
basal_profile = settings/basal_profile.json
reporter = text
json_default = True
max_iob = preferences.json
device = get-profile
remainder =
insulin_sensitivities = settings/insulin_sensitivities.json
```

### 4.3.6 Wifi and hotspot issues

See wifi troubleshooting page

## 4.4 OpenAPS Overview and Project History

In order to relieve the incredible burden of T1D, many research teams and manufacturers have developed and are testing Artificial Pancreas Systems (APSs) that connect CGMs to insulin pumps and use various algorithms to automatically adjust insulin dosing (and sometimes dose glucagon, a counter-regulatory hormone) to attempt to mimic some of the functions of a healthy pancreas, and keep blood sugar levels in a safe range. While quite successful in clinical trials so far, current APS systems have been in development for many years, and are still likely at least 3 years away from FDA approval. It is also unclear whether first-generation APS technology will be suitable for, or available to, all patients, even in rich countries.

To address some of the challenges of daily life with diabetes, and because #WeAreNotWaiting, several people worked to figure out how to connect up existing FDA-approved medical devices such as the Dexcom G4 CGM and the Medtronic Minimed insulin pump, using commodity computer / mobile phone hardware and open-source software, to create a complete closed loop Artificial Pancreas System (APS). The first public example of this was the #DIYPS closed loop system, created in their spare time by @DanaMLewis and @ScottLeibrand in the fall of 2013 based on their earlier work to build the #DIYPS remote monitoring and decision assist system. #DIYPS used the Nightscout project's uploader to get Dexcom CGM data off the device. #DIYPS was able to become a closed loop with the help of open-source decoding-carelink project created by @Ben West to communicate with Medtronic insulin pumps, retrieve data and issue insulin-dosing commands to pumps that support it. #DIYPS was the base system that led to #OpenAPS.

In light of the success of #DIYPS closed loop and other simple APS systems built by individuals, Dana and Scott decided to further apply the #WeAreNotWaiting ethos to APS research, believing safe and effective APS technology can be made available more quickly and to more people, rather than just waiting for current APS efforts to complete clinical trials and be FDA-approved and commercialized through traditional processes.

#OpenAPS is an open reference design for, and will be a reference implementation of, an overnight closed loop APS system that uses the CGM sensors' estimate of blood glucose (BG) to automatically adjust basal insulin levels, in order to keep BG levels inside a safe range overnight and between meals.

#OpenAPS is not intended to be a "set and forget" APS system. To maximize safety, a system designed from OpenAPS only doses basal insulin. Users still need to bolus for meals as they do today. However, OpenAPS can identify deviations from predicted blood sugar changes and change basal rates to prevent dangerous drops or rises that deviate from expected behavior.

After launching in early 2015, there are at least 44 known instances of OpenAPS that are live and running (as of March 23, 2016), with several others in development and testing phases. For anecdotal experiences from those running OpenAPS, watch the #OpenAPS hashtag on Twitter and also check out the Resources section for a list of those sharing their experiences publicly.

In early 2016, progress continues to be made with the iteration of several hardware options, in addition to multiple new software features.

## 4.5 Other People, Projects & Tools

### 4.5.1 People

These people have publicly identified as either supporting or running OpenAPS implementations and are sharing their experiences publicly. See below links.

Dana Lewis - blogs about personal experience at DIYPS.org and shares on Twitter as @DanaMLewis.

Scott Leibrand - contributes to Dana's instance of OpenAPS, and is on Twitter as @ScottLeibrand. (Scott and Dana collectively maintain OpenAPS.org.)

Ali Mazaheri - shares on Twitter as @AliMazaheri

Chris Hannemann - shares on Twitter as @hannemannemann

Nate Racklyeft - shares on Twitter as @LoudNate

Ben West - author of decoding-carelink and much of the openaps toolkit - on Twitter as @bewestisdoing

The following provide links to other related projects as well as commercial artificial pancreas work underway.

### 4.5.2 APS & Diabetes Data Tools

- **#DIYPS** (http://diyps.org/) - the project and personal experience that inspired #OpenAPS
- **simPancreas** (http://bustavo.com/category/simpancreas/) - another DIY closed loop, although not #OpenAPS related
- **NightScout** (http://www.nightscout.info/) - a visualization and remote monitoring tool for people with diabetes using CGM
- **xDrip** (http://stephenblackwasalreadytaken.github.io/xDrip/) - a DIY combination of a device and a software application which receives data sent out by a Dexcom G4 CGM transmitter/sensor and displays the glucose readings on an Android phone
- **RileyLink** (https://github.com/ps2/rileylink) A custom designed Bluetooth Smart (BLE) to 916MHz module. It can be used to bridge any BLE capable smartphone to the world of 916Mhz based devices. This project is focused on talking to Medtronic insulin pumps and sensors. There is also a Gitter channel dedicated to discussion on the RileyLink here.
- **Tidepool** (http://tidepool.org/ and https://github.com/tidepool-org) Notably, work on Boston University iLet UI (https://github.com/tidepool-org/bionicpancreas) and open-source tools for visualization.
- **Perceptus** (http://perceptus.org) - more data visualization tools

## 4.6 Commercial APS Efforts

There are currently several commercial closed-loop products in development by old and new companies in the diabetes treatment space. These include:

- Medtronic MiniMed 640G
- Medtronic MiniMed 670G
- TypeZero Technologies
- Bigfoot Biomedical
- Boston University's iLet, formerly known as the "Bionic Pancreas"

## 4.7 Frequently Asked Questions

### 4.7.1 What is a Closed Loop?

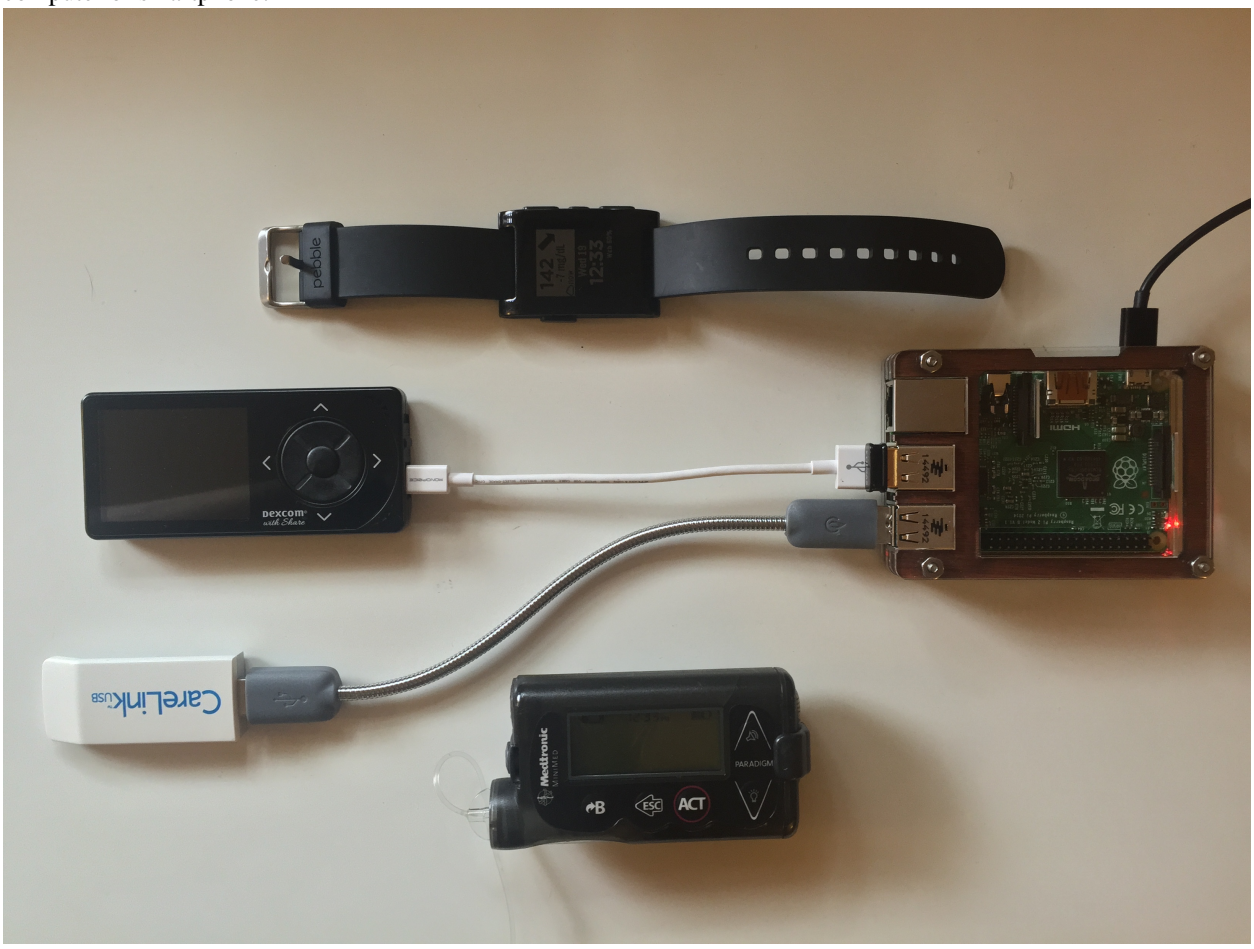
In general, a "closed loop" system for treatment of diabetes is considered to be one in which insulin dosing "and in some cases glucagon dosing" is partially or completely automated. This is in contrast to an "open loop" system, where the user evaluates the inputs and manually instructs the insulin pump to dose a specific amount. In both cases, the goal is to maintain blood glucose within the desired range through adjusting hormone doses.

There are numerous different types of closed loop systems, ranging from simple basal suspend systems designed to mitigate extreme hypoglycemia to dual hormone, fully automated systems. The JDRF Artificial Pancreas Project Plan page provides an overview of the current commercial and academic generation-based approach. Several commercial systems are currently in development; see Commercial APS Efforts for more information.

#OpenAPS is focused on a single-hormone hybrid closed-loop system. This is a system that uses only insulin (no glucagon) and still requires user input for mealtime insulin. For background on #OpenAPS, review the #OpenAPS Reference Design page.

### 4.7.2 What does an OpenAPS closed loop look like?

While there are numerous variations, this particular setup shows the key components—namely, a continuous glucose monitor, an insulin pump, a method for communicating with the pump (here, a CareLink USB stick), and a controller (here, a Raspberry Pi). Also shown is a Pebble watch, which can be used for monitoring the status of the OpenAPS. Not shown is the power supply (off-screen) and a way to interact with and program the Raspberry Pi, typically a computer or smartphone.




For more details on the exact hardware required to build an OpenAPS, see the Hardware section.

## 4.8 Glossary

APS - artificial pancreas system. Sometimes also referred to as “AP”

CGM - continuous glucose monitor, a temporary glucose sensor that is injected into your skin (the needle is removed) for 3-7 days and, with twice a day calibrations, provides BG readings approximately every 5 minutes.

#OpenAPS - stands for Open A(rtificial) P(ancreas) S(ystem). It is an open-source movement to develop an artificial pancreas using commercial medical devices, a few pieces of inexpensive hardware, and freely-available software. A full description of the #OpenAPS project can be found at openaps.org. #OpenAPS (with the hashtag) generally refers to the broad project and open source movement.

OpenAPS - refers to an example build of the system when used without a hashtag (#)

openaps - the core suite of software tools under development by this community for use in an OpenAPS implementation

Bolus - extra insulin given by a pump, usually to correct for a high BG or for carbohydrates

Basal - baseline insulin level that is pre-programmed into your pump and mimics the insulin your pancreas would give throughout the day and night

IOB - Insulin On Board, or insulin active in your body. Note that most commercially available pumps calculate IOB based on bolus activity only. An OpenAPS implementation calculates and refers most often to net IOB, which takes into account any adjusted (higher or lower) basal rates as well as bolus activity.

DIA - duration of insulin action, or how long the insulin is active in your body. (Ranges 3-6 hours typically)

CR - carb ratio, or carbohydrate ratio - the amount of carbohydrates for one unit of insulin. Example: 1 u of insulin for 10 carbs

ISF - insulin sensitivity factor - the amount of insulin that drops your BG by a certain amount mg/dl. Example: 1 u of insulin for 40 mg/dl

NS, or Nightscout - a cloud-based visualization and remote-monitoring tool.

CHAPTER 5

# Reference

## 5.1 Manuals

### 5.1.1 openaps

#### openaps

##### Help

usage: openaps [-h] [-c C C] [-C CONFIG] [–version] [command] ...

openaps - openaps: a toolkit for DIY artificial pancreas system

positional arguments: command args

optional arguments: -h, –help show this help message and exit -c C C -C CONFIG, –config CONFIG –version show program's version number and exit

Utilities for developing an artificial pancreas system. openaps helps you manage and structure reports for various devices.

```
   .-. .,-.   .-. .--. .-.   .,-. .--.
  (   )|   )(.-' |  |(   )  |   )`--.
   `-' |`-'  `--''  `-`-'`-|`-' `--'
       |                  |
       '                  '
```

Common workflows:

Getting started:

openaps init - create a new instance of openaps openaps init myopenaps - this creates an instance of openaps in a new directory, called myopenaps

cd myopenaps - change directory to root of new repo

A valid instance of openaps is a git repo with a file called openaps.ini present.

openaps will track configuration and some status information inside of openaps.ini. If you already have a git repo which you would like to become a valid openaps environent, in the root of your repo, run:

```
touch openaps.ini
git add openaps.ini
git commit -avm 'init openaps'
```

Now, wth a valid openaps environment, you can register devices for use. A device is implemented by a vendor. openaps [will] provide a modular, language and process independent environment for creating vendors and devices, but for now the only two are dexcom and medtronic.

To register devices for use, see: openaps device -h openaps device add [opts...] eg:

#### register a medtronic device named pump

openaps device add pump medtronic 665455

#### register a dexcom device named cgm

openaps device add cgm dexcom

Now that devices are known, and we have a variety of commands available. We can explore how to produce reports by using devices with the openaps use command:

```
openaps use <device-name> <use-name> [opts]
```

openaps use commands can only be used after devices have been added to the openaps.ini config using openaps device add. Eg: openaps use pump -h - show available commands for the device known as "pump" openaps use pump iter_pump - get last 100 pump history records from the device called pump openaps use cgm -h - show available commands for the device known as "cgm" openaps use cgm glucose

After experimenting with openaps use commands, users can save reports using the openaps report commands. openaps report commands map openaps use commands to filenames:

```
openaps report add <report-name> <report-formatter> <device> <use> [opts]

# add a report, saved in a file called pump-history.json, which is
# JSON format, from device pump using use iter_pump.
openaps report add pump-history.json JSON pump iter_pump

# add a report, saved in a file called glucose.json, which is
# JSON format, from device cgm using use glucose.
openaps report add glucose.json JSON cgm glucose

# invoke the report to create glucose.json
openaps report invoke glucose.json

# invoke the report to create pump-history.json
openaps report invoke pump-history.json
```

All commands support tab completion, and -h help options to help explore the live help system.

### openaps-alias

#### Help

usage: openaps-alias [-h] {add,remove,show} ...

openaps-alias - manage aliases

optional arguments: -h, –help show this help message and exit

#### Alias Menu:

aliases - manage alias configurations

{add,remove,show} Operation add add - add an alias remove remove - remove an alias show show - show all aliases

### openaps-dbus

#### Help

usage: openaps-dbus [-h] [–quit] [–name NAME] [–ini_home INI_HOME] [–create CREATE]

optional arguments: -h, –help show this help message and exit –quit –name NAME –ini_home INI_HOME –create CREATE

### cgm.ini

This is the device file for your Dexcom CGM. It will generally not have anything in it, as the communications are handled by core OpenAPS code.

#### Setup code

```
openaps device add cgm dexcom
```

#### Sample contents

```
[device "cgm"]
```

#### Dependencies

None

### pump.ini

This device contains the information requied to connect to your pump, such as your pump's serial number. This example assumes you are using a TI stick.

#### Setup code

```
openaps device add pump mmeowlink subg_rfspy /dev/mmeowlink [YOUR PUMP SERIAL]
```

#### Sample contents

‘ [device “pump”] serial = [YOUR PUMP SERIAL] port = /dev/mmeowlink radio_type = subg_rfspy model = [YOUR PUMP MODEL] expires = 2016-05-23T23:02:36.168762 ‘

#### Dependencies

None

### openaps-enact

#### Help

usage: openaps-enact [-h] [–version] [{temp-basal,bolus,phone,sms,pipe}] [{simulator,static-file,medtronic,dexcom}] ...

Send changes out into the world.

positional arguments: {temp-basal,bolus,phone,sms,pipe} {simulator,static-file,medtronic,dexcom} args

optional arguments: -h, –help show this help message and exit –version show program's version number and exit

Make a phone call, send SMS/TEXT, send messages to devices and people.

### openaps-get

#### Help

usage: openaps-get [-h] [–version] [{glucose,basals,help,pump}] [{simulator,static-file,medtronic,dexcom}] ...

positional arguments: {glucose,basals,help,pump} {simulator,static-file,medtronic,dexcom} args

optional arguments: -h, –help show this help message and exit –version show program's version number and exit

### openaps-import

#### Help

usage: openaps-import [-h] [–list] [input]

openaps-import - import openaps configuration Import configuration.

positional arguments: input Read from stdin

optional arguments: -h, –help show this help message and exit –list, -l Just list types

Example workflow:

openaps -C /path/to/another/openaps.ini device show –json ns-upload | openaps import -

### openaps-install-udev-rules

#### Help

run as root

### openaps-report

#### Help

usage: openaps-report [-h] [–version] {add,remove,show,invoke} ...

openaps-report - configure reports

optional arguments: -h, –help show this help message and exit –version show program's version number and exit

#### Reports Menu:

reports - manage report configurations

{add,remove,show,invoke} Operation add add - add a new report configuration remove remove - remove a device configuration show show - show all reports invoke invoke - generate a report

Manage which devices produce which reports.

Example workflow:

Use the add, remove, show to manage which reports openaps knows about.

The add command adds a new report to the system. The syntax is: add

```
openaps report add my-results.json json pump basals

This example registers a json output, using the pump basals command, and
stores the result in my-results.json.
```

The show command will list or give more details about the reports registered with openaps. The syntax is: show [name]. The default name is '*' which should list all available reports.

```
openaps report show
```

The remove command removes the previously configured report from openaps. The syntax is: remove

```
openaps report remove my-results.json
This example removes the report "my-results.json" from the openaps
environment.
```

openaps report invoke basals

### max_iob.json

This is less of a report, and more of a file created during setup.

#### Setup code

None

#### Sample contents

```
{ "max_iob":  0 }
```

#### Dependencies

None

### monitor/battery.json

This report contains the current status and voltage of the battery in your pump.

#### Setup code

```
openaps report add monitor/battery.json JSON pump read_battery_status
```

#### Sample contents

```
{ "status":  "normal", "voltage":  1.56 }
```

#### Dependencies

None

### monitor/clock.json

This report contains the date and time that is set on your pump, but does **NOT** include timezone information.

#### Setup code

```
openaps report add monitor/clock.json JSON pump read_clock
```

#### Sample contents

```
"2016-05-22T00:18:41"
```

#### Dependencies

- pump.ini

### monitor/clock-zoned.json

This report contains the date and time that is set on your pump, but modified to include your timezone information.

#### Setup code

```
openaps use tz clock --timezone "[YOUR TIMEZONE]" --adjust "missing" --date
"None" --astimezone monitor/clock.json
```

#### Sample contents

```
"2016-05-23T22:40:14-04:00"
```

#### Dependencies

- monitor/clock.json

### monitor/glucose.json

This report contains multiple glucose entries from either your CGM or Nightscout install. For simplicity's sake, this document assumes that the data is coming from your directly connected CGM.

#### Setup code

```
openaps report add monitor/glucose.json JSON cgm oref0_glucose --hours "25.0"
--threshold "100"
```

#### Sample contents

```
[ { "trend_arrow": "FLAT",. "display_time": "2016-05-22T00:22:27",.
"direction": "Flat",. "system_time": "2016-05-22T07:22:27",. "sgv": 149,.
"dateString": "2016-05-22T00:22:27-04:00",. "device": "openaps://cgm",.
"unfiltered": 159840,. "rssi": 180,. "date": 1463890947000.0,.
"filtered": 156928,. "type": "sgv",. "glucose": 149 }, ]
```

#### Dependencies

- cgm.ini

### monitor/iob.json

This report contains several entries detailing the levels of IOB (Insulin On Board) over a given time period.

#### Setup code

```
openaps report add monitor/iob.json text iob shell monitor/pumphistory-zoned.json
settings/profile.json monitor/clock-zoned.json
```

#### Sample contents

```
[ { "iob": 1.908, "activity": 0.0009, "bolussnooze": 0, "basaliob":
1.908, "netbasalinsulin": 1.1, "hightempinsulin": 3.2, "time":
"2016-05-22T04:43:33.000Z" }, ]
```

#### Dependencies

- monitor/pumphistory-zoned.json
- settings/profile.json
- monitor/clock-zoned.json

### monitor/pumphistory.json

This report gathers the last 5 hours of history directly from your pump.

#### Setup code

```
openaps report add monitor/pumphistory.json JSON pump iter_pump_hours 5
```

#### Sample contents


```
[ { "_type":  "TempBasalDuration",.  "duration (min)":  30,.  "_description":
"TempBasalDuration 2016-05-23T22:15:28 head[2], body[0] op[0x16]",.
"timestamp":  "2016-05-23T22:15:28",.  "_body":  "",.  "_head":  "1601",.
"_date":  "5c4f165710" }, ]
```


#### Dependencies

- pump.ini

### monitor/pumphistory-zoned.json

This report is the same as your `pumphistory.json` report, but adjusted for your timezone.

#### Setup code

```
openaps use tz rezone --timezone "[YOUR TIMEZONE]" --adjust "missing"
--date "timestamp dateString start_at end_at created_at" --astimezone
monitor/pumphistory.json
```

#### Sample contents


```
[ { "_type":  "TempBasalDuration",.  "_description":  "TempBasalDuration
2016-05-23T22:15:28 head[2], body[0] op[0x16]",.  "timestamp":
"2016-05-23T22:15:28-04:00",.  "_body":  "",.  "_head":  "1601",.  "duration
(min)":  30,.  "_date":  "5c4f165710" }, ]
```


#### Dependencies

- monitor/pumphistory.json

### settings/auto-sens.json

This report contains an automatically determined, temporary modification to your ISF (Insulin Sensitivity Factor).

#### Setup code

```
openaps report add settings/auto-sens.json text detect-sensitivity
shell monitor/glucose.json settings/pumphistory-24h-zoned.json
settings/insulin_sensitivities.json settings/basal_profile.json
settings/profile.json
```

#### Sample contents

```
{"ratio":0.78}
```

#### Dependencies

- monitor/glucose.json
- settings/pumphistory-24h-zoned.json
- settings/insulin_sensitivities.json
- settings/basal_profile.json
- settings/profile.json

### settings/basal_profile.json

This report contains the basal rates that are set up in your pump.

#### Setup code

```
openaps report add settings/basal_profile.json JSON pump read_selected_basal_profile
```

#### Sample contents

```
[ { "i":  0, "start":  "00:00:00", "rate":  1.8, "minutes":  0 }, { "i":  1,
"start":  "05:00:00", "rate":  2.2, "minutes":  300 }, { "i":  2, "start":
"08:00:00", "rate":  1.8, "minutes":  480 } ]
```

#### Dependencies

- pump.ini

### settings/bg_targets.json

This report contains the high/low glucose targets set up in your pump. OpenAPS has a hardcoded "min_bg" floor of 90, which will override any pump low target bg value below 90.

#### Setup code

```
openaps report add settings/bg_targets.json JSON pump read_bg_targets
```

#### Sample contents

```
{ "units": "mg/dL", "raw": "0x01 0x00 0x64 0x78 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00", "targets": [ { "high":
120, "start": "00:00:00", "low": 100, "offset": 0, "i": 0, "x": 0 } ],
"first": 1 }
```

#### Dependencies

- pump.ini

### settings/carb_ratios.json

This report contains your carb ratios.

#### Setup code

```
openaps report add settings/carb-ratios.json JSON pump read_carb_ratios
```

#### Sample contents

```
{ "units": "grams", "raw": "0x01 0x00 0x05 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00", "first": 1, "schedule":
[ { "start": "00:00:00", "r": 5, "ratio": 5, "offset": 0, "i": 0, "x": 0
} ] }
```

#### Dependencies

- pump.ini

### settings/insulin_sensitivities.json

This report contains the insulin sensitivity levels stored in your pump.

#### Setup code

```
openaps report add settings/insulin_sensitivities.json JSON pump
read_insulin_sensitivities
```

#### Sample contents

```
{ "units":  "mg/dL", "sensitivities":  [ { "i":  0, "start":  "00:00:00",
"sensitivity":  20, "offset":  0, "x":  0 } ], "first":  1 }
```

#### Dependencies

- pump.ini

### settings/model.json

This report contains the model number of your pump.

#### Setup code

```
openaps report add settings/model.json JSON pump model
```

#### Sample contents

```
"722"
```

#### Dependencies

- pump.ini

### settings/profile.json

This report contains a general profile of the information in your pump such as carb ratios, DIA, max basal rates, etc.

#### Setup code

```
openaps report add settings/profile.json text get-profile shell
settings/settings.json settings/bg_targets.json settings/insulin_sensitivities.json
settings/basal_profile.json max_iob.json
```

#### Sample contents

```
{"max_iob":0,"type":"current","dia":4,"current_basal":1.8,"max_daily_basal":2.2,"max_basal"
```

#### Dependencies

- settings/settings.json
- settings/bg_targets.json
- settings/insulin_sensitivities.json
- settings/basal_profile.json
- max_iob.json

### settings/settings.json

This report contains various settings specific to your pump.

#### Setup code

```
openaps report add settings/settings.json JSON pump read_settings
```

#### Sample contents

```
{ "low_reservoir_warn_point": 20, "keypad_lock_status": 0,
"maxBasal": 6.0, "temp_basal": { "percent": 100, "type": "Units/hour"
}, "low_reservoir_warn_type": 0, "insulinConcentration": 100,
"audio_bolus_enable": true, "variable_bolus_enable": true, "alarm": {
"volume": 2, "mode": 2 }, "rf_enable": false, "auto_off_duration_hrs":
24, "block_enable": false, "timeformat": 0, "insulin_action_curve": 4,
"audio_bolus_size": 1.0, "selected_pattern": 0, "patterns_enabled": true,
"maxBolus": 25.0, "paradigm_enabled": 1 }
```

#### Dependencies

- pump.ini

### openaps-schedule

#### Help

usage: openaps-schedule [-h] {add,show,remove} ...

openaps-schedules - manage schedules

optional arguments: -h, –help show this help message and exit

#### Schedule management:

schedules - manage schedules and triggers

{add,show,remove} Operation add add - add a scheduled event show show - show all schedules remove remove - remove a schedule

### openaps-trigger

#### Help

usage: openaps-trigger [-h] {add,show,remove,emit,until} ...

openaps-triggers - manage triggers

optional arguments: -h, –help show this help message and exit

#### Trigger management:

triggers - manage triggers and triggers

{add,show,remove,emit,until} Operation add add - add a managed Trigger show show - show all trigger remove remove - remove a trigger emit emit - emit a trigger's events until until - wait until a trigger emits events

### openaps-use

#### Help

usage: openaps-use [-h] [–format {text,json,base,stdout}] [–output OUTPUT] [–version] device ...

openaps-use - use a registered device

optional arguments: -h, –help show this help message and exit –format {text,json,base,stdout} –output OUTPUT –version show program's version number and exit

Known Devices Menu: These are the devices openaps knows about:

device Name and description: black Medtronic - openaps driver for Medtronic blue Medtronic - openaps driver for Medtronic calculate-iob process - a fake vendor to run arbitrary commands cat process - a fake vendor to run arbitrary commands cgm Dexcom - openaps driver for dexcom curl process - a fake vendor to run arbitrary commands determine-basal process - a fake vendor to run arbitrary commands dx-format-oref0-glucose process - a fake vendor to run arbitrary commands format-latest-nightscout-treatments process - a fake vendor to run arbitrary commands get-profile process - a fake vendor to run arbitrary commands howdy process - a fake vendor to run arbitrary commands iob process - a fake vendor to run arbitrary commands latest-treatments process - a fake vendor to run arbitrary commands mine process - a fake vendor to run arbitrary commands munge mmhistorytools - tools for cleaning, condensing, and reformatting history data my-agp AGP - calculate agp values given some glucose text newone process - a fake vendor to run arbitrary commands ns-glucose process - a fake vendor to run arbitrary commands ns-upload process - a fake vendor to run arbitrary commands oref0 process - a fake vendor to run arbitrary commands plugins Plugins - Community maintained plugins to openaps predict predict - tools for predicting glucose trends pump mmeowlink - openaps driver for cc1111/cc1110 devices share openxshareble - pure python driver to communicate with Dexcom G4+Share over ble. Allows openaps to use ble to talk to Dexcom G4+Share. tz Timezones - manage timezones in diabetes data with ease. units Units - units tool for openaps

Once a device is registered in openaps.ini, it can be used.

### openaps-vendor

#### Help

usage: openaps-vendor [-h] {add,remove,show} ...

openaps-vendor - Manage vendor plugins.

optional arguments: -h, –help show this help message and exit

#### Vendors Menu:

vendors - manage vendor plugin configurations

{add,remove,show} Operation add Add a new vendor plugin to openaps-environment. remove Remove vendor plugin from openaps-environment show Show/list vendor plugins

show - lists all known vendors add - add a new vendor remove - remove a vendor

### 5.1.2 oref0

#### oref0

#### Help

Usage: oref0

```
 _______   _______   _______  _______ 0
/ |   | \ | |   | \ | |      | |
| |   | | | |__| | | | |----- | |----
\_|__|_/ |_|   \_\ |_|_____ |_|
```

Valid commands: oref0 env - print information about environment. oref0 pebble oref0 ifttt-notify oref0 get-profile oref0 calculate-iob oref0 meal oref0 determine-basal oref0 help - this message

#### oref0-calculate-iob

#### Help

usage: [ ‘node’, ‘/usr/local/bin/oref0-calculate-iob’ ] <pumphistory.json> <profile.json> <clock.json>

#### oref0-determine-basal

#### Help

#### oref0-find-insulin-uses

#### Help

usage: [ ‘node’, ‘/usr/local/bin/oref0-find-insulin-uses’ ] <pumphistory.json> <profile.json>

#### oref0-fix-git-corruption

#### Help

Git repo does not appear to be corrupt.

### oref0-get-profile

#### Help

usage: [ 'node', '/usr/local/bin/oref0-get-profile' ] <pump_settings.json> <bg_targets.json> <insulin_sensitivities.json> <basal_profile.json> [<preferences.json>] [<carb_ratios.json>]

### oref0-ifttt-notify

#### Help

Usage: oref0-ifttt-notify <IFTTT_TRIGGER> <NOTIFY_USAGE>

#### Setup IFTTT Account

You need to create an account and connect to the Maker channel and the notification channel of your choice. I use pushover as I already had the app and it allows me more control over the notification on my phone.

#### Create an IF recipe

The trigger is the maker channel. You can customize the notification message if you wish.

#### Get the event trigger

On the Maker channel there is a "how to trigger" link. Copy and paste the url for the example curl command, be sure to change the event name field. The URL, something like:

```
https://maker.ifttt.com/trigger/{event}/with/key/MyKey
```

You can pass the IFTTT_TRIGGER, which is the trigger URL as the first argument, or define it as an environment variable in your crontab.

Command line:

```
  oref0-ifttt-notify https://maker.ifttt.com/trigger/{event}/with/key/MyKey
```

Crontab: IFTTT_TRIGGER=https://maker.ifttt.com/trigger/{event}/with/key/MyKey

By default oref0-ifttt-notify will check that the stick works, and notify the IFTTT_TRIGGER endpoint only if the the stick fails to check out. If the stick diagnostics indicate the carelink stick is working, oref0-ifttt-notify will run:

```
  openaps use pump model
```

You can specify which openaps use command to use in the second argument, or by setting the IFTTT_NOTIFY_USAGE environment variable in crontab:

Command line, note the quotes, the second term must be passed as single word.

```
  oref0-ifttt-notify https://maker.ifttt.com/trigger/{event}/with/key/MyKey 'pump model'
```

Crontab:

```
  IFTTT_NOTIFY_USAGE='pump model'
```

Author: @audiefile

### oref0-meal

#### Help

usage: [ 'node', '/usr/local/bin/oref0-meal' ] <pumphistory.json> <profile.json> <clock.json> [carbhistory.json]

### oref0-mint-max-iob

#### Help

Usage: oref0-mint-max-iob <max_iob> [preferences.json]

oref0-mint-max-iob help - this message Print a perfect preferences.json.

Examples: $ oref0-mint-max-iob 2 { "max_iob": 2 }

$ oref0-mint-max-iob 2 foo.json max_iob 2 saved in foo.json

### oref0-normalize-temps

#### Help

usage: [ 'node', '/usr/local/bin/oref0-normalize-temps' ] <pumphistory.json>

### oref0-pebble

#### Help

usage: [ 'node', '/usr/local/bin/oref0-pebble' ] <glucose.json> <iob.json> <current_basal_profile.json> <current-temp.json> <requestedtemp.json> <enactedtemp.json> [meal.json]

### oref0-raw

#### Help

### oref0-reset-git

#### Help

oref0-reset-git oref0-reset-git - Wipe out all history, forcibly re-initialzize openaps from scratch.

### oref0-reset-usb

#### Help

oref0-reset-usb oref0-reset-usb - Drop USB stack, rebind the usb kernel modules.

### 5.1.3 Nightscout

#### `nightscout`

##### Help

Usage: nightscout

- latest-openaps-treatment
- cull-latest-openaps-treatments
- ns-get
- ns-upload
- ns-dedupe-treatments
- ns-status
- ns-upload-entries

#### `ns-dedupe-treatments`

##### Help

Usage: ns-dedupe-treatments –find <NIGHTSCOUT_HOST> - No-op version, find out what delete would do. ns-dedupe-treatments –list <NIGHTSCOUT_HOST> - list duplicate count per created_at ns-dedupe-treatments delete <NIGHTSCOUT_HOST> - Delete duplicate entries from <NIGHTSCOUT_HOST>

#### `ns-status`

##### Help

usage: [ 'node', '/usr/local/bin/ns-status' ] <clock.json> <iob.json> <suggested.json> <enacted.json> <battery.json> <reservoir.json> <status.json> [mmtune.json]

#### `ns-upload`

##### Help

ns-upload: missing API_SECRET

#### `ns-upload-entries`

##### Help

ns-upload-entries <entries.json> http://nightscout.host:1337 ns-upload-entries - Upload entries (glucose data) to NS.

### 5.1.4 mm

#### mm-bolus.py

##### Help

usage: mm-bolus.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] (–515 | –554 | –strokes STROKES_PER_UNIT) units

mm-bolus.py - Send bolus command to a pump.

positional arguments: units Amount of insulin to bolus.

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –515 –554 –strokes STROKES_PER_UNIT

XXX: Be careful please! Units might be wrong. Keep disconnected from pump until you trust it by observing the right amount first.

#### mm-decode-history-page.py

##### Help

usage: mm-decode-history-page.py [-h] [–collate] [–data {glucose,pump}] [–model {740,751,522,754,554,715,530,551,730,540,722,723,508,523,511,512,515}] [–larger] [–no-larger] [–out OUT] infile [infile ...]

positional arguments: infile Find dates in this file.

optional arguments: -h, –help show this help message and exit –collate –data {glucose,pump} –model {740,751,522,754,554,715,530,551,730,540,722,723,508,523,511,512,515} –larger –no-larger –out OUT Write records here.

#### mm-format-ns-glucose

##### Help

mm-format-ns-glucose [–oref0] <medtronic-glucose.json> mm-format-ns-glucose - Format Medtronic glucose data into something acceptable to Nightscout.

#### mm-format-ns-profile

##### Help

mm-format-ns-profile: Format known pump data into Nightscout "profile".

Profile documents allow Nightscout to establish a common set of settings for therapy, including the type of units used, the timezone, carb-ratios, active basal profiles, insulin sensitivities, and BG targets. This compiles the separate pump reports into a single profile document for Nightscout.

Usage: mm-format-ns-profile pump-settings carb-ratios active-basal-profile insulin-sensitivities bg-targets

Examples: bewest@bewest-MacBookPro:~/Documents/openaps$ mm-format-ns-profile monitor/settings.json monitor/carb-ratios.json monitor/active-basal-profile.json monitor/insulin-sensitivities.json monitor/bg-targets.json

### mm-format-ns-pump-history

#### Help

mm-format-ns-pump-history <medtronic-pump-history.json> mm-format-ns-pump-history - Format Medtronic pump-history data into something acceptable to Nightscout.

### mm-format-ns-treatments

#### Help

mm-format-ns-treatments <pump-history-zoned.json> <model.json> mm-format-ns-treatments - Format medtronic history data into Nightscout treatments data.

### mm-latest.py

#### Help

usage: mm-latest.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–no-clock] [–no-basal] [–no-temp] [–no-reservoir] [–no-status] [–parser-out PARSED_DATA] [–rtc-out RTC_ARCHIVE] [–reservoir-out RESERVOIR_ARCHIVE] [–settings-out SETTINGS] [–temp-basal-status-out TEMPBASAL] [–basals-out BASALS] [–status-out STATUS] [–timezone TIMEZONE] [minutes]

mm-latest.py - Grab latest activity

positional arguments: minutes [default: 30)]

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don’t do main function –skip-prelude Don’t do the normal prelude. –no-rf-prelude Do the prelude, but don’t query the pump. –skip-postlude Don’t do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –no-clock Also report current time on pump. –no-basal Also report basal rates. –no-temp Also report temp basal rates. –no-reservoir Also report remaining insulin in reservoir. –no-status Also report current suspend/bolus status. –parser-out PARSED_DATA Put history json in this file –rtc-out RTC_ARCHIVE Put clock json in this file –reservoir-out RESERVOIR_ARCHIVE Put reservoir json in this file –settings-out SETTINGS Put settings json in this file –temp-basal-status-out TEMPBASAL Put temp basal status json in this file –basals-out BASALS Put basal schedules json in this file –status-out STATUS Put status json in this file –timezone TIMEZONE Timezone to use

Query pump for latest activity.

### mm-press-key.py

#### Help

usage: mm-press-key.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] {act,esc,up,down,easy} [{act,esc,up,down,easy} ...]

mm-press-key.py - Simulate presses on the keypad.

positional arguments: {act,esc,up,down,easy} buttons to press [default: None)]

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session.

Press keys on the keypad.

### mm-pretty-csv

#### Help

### mm-send-comm.py

#### Help

usage: mm-send-comm.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–prefix-path PREFIX_PATH] [–saveall] [–prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc [–postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,Guardia {sleep,tweak,ManualCommand} ...

mm-send-comm.py - send messages to a compatible MM insulin pump

positional arguments: {sleep,tweak,ManualCommand} Main thing to do between –prefix and–postfix sleep Just sleep between command sets tweak Tweak a builtin command ManualCommand Customize a command

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –prefix-path PREFIX_PATH Prefix to store saved files when using –save or –saveall. –saveall Whether or not to save all responses. –prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc Built-in commands to run before the main one. –postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc Built-in commands to run after the main one.

This tool is intended to help discover protocol behavior. Under no circumstance is it intended to deliver therapy.

### mm-set-rtc.py

#### Help

usage: mm-set-rtc.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–rtc-out RTC_ARCHIVE] [–timezone TIMEZONE] –set SET [–out OUT] {query,set}

mm-set-rtc.py - query or set RTC

positional arguments: {query,set} Set or query pump status [default: query)]

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –rtc-out RTC_ARCHIVE Put clock json in this file –timezone TIMEZONE Timezone to use –set SET Set clock to new value (iso8601) –out OUT Put basal in this file

Set or query RTC.

#### mm-set-suspend.py

##### Help

usage: mm-set-suspend.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] {query,suspend,resume} [{query,suspend,resume} ...]

mm-set-suspend.py - query or set suspend/resume status

positional arguments: {query,suspend,resume} Set or query pump status [default: query)]

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session.

Pause or resume pump.

#### mm-stick

##### Help

Usage: mm-stick [{scan,diagnose,help},...]

```
scan      - Print the local location of a plugged in stick.
diagnose  - Run python -m decocare.stick $(python -m decocare.scan)
warmup    - Runs scan and diagnose with no output.
            Exits 0 on success, non-zero exit code
            otherwise.
insert    - Insert usbserial kernel module.
remove    - Remove usbserial kernel module.
udev-info - Print udev information about the stick.
list-usb  - List usb information about the stick.
reset-usb - Reset entire usb stack. WARNING, be careful.
fail      - Always return a failing exit code.
help      - This message.
```

#### mm-temp-basals.py

##### Help

usage: mm-temp-basals.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–duration DURATION] [–rate RATE] [–out OUT] {query,set,percent}

mm-temp-basals.py - query or set temp basals

positional arguments: {query,set,percent} Set or query pump status [default: query)]

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –duration DURATION Duration of temp rate [default: 0)] –rate RATE Rate of temp basal [default: 0)] –out OUT Put basal in this file

Set or query temp basals.

### mmblelink

#### Help

### mmblelink-send.py

#### Help

usage: mmblelink-send.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–rx {0,1,2,0,1,2,PumpTX,PumpRX}] [–tx {0,1,2,0,1,2,PumpTX,PumpRX}] [–sleep_interval SLEEP_INTERVAL] [–prefix-path PREFIX_PATH] [–saveall] [–prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc [–postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,Guardia MAC {sleep,tweak,ManualCommand} ...

mmblelink adapter to decocare's SendMsgApp

positional arguments: MAC RileyLink address {sleep,tweak,ManualCommand} Main thing to do between –prefix and–postfix sleep Just sleep between command sets tweak Tweak a builtin command ManualCommand Customize a command

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –rx {0,1,2,0,1,2,PumpTX,PumpRX}, -R {0,1,2,0,1,2,PumpTX,PumpRX} –tx {0,1,2,0,1,2,PumpTX,PumpRX}, -T {0,1,2,0,1,2,PumpTX,PumpRX} –sleep_interval SLEEP_INTERVAL, -s SLEEP_INTERVAL Amount to sleep between polling. –prefix-path PREFIX_PATH Prefix to store saved files when using –save or –saveall. –saveall Whether or not to save all responses. –prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc Built-in commands to run before the main one. –postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc Built-in commands to run after the main one.

### mmcli

#### Help

Usage: mmcli [OPTION...] - Control and monitor the ModemManager

Help Options: -h, –help Show help options –help-all Show all help options –help-manager Show manager options –help-common Show common options –help-modem Show modem options –help-3gpp Show 3GPP related options –help-cdma Show CDMA related options –help-simple Show Simple options –help-location Show Location options –help-messaging Show Messaging options –help-time Show Time options –help-firmware Show Firmware options –help-sim Show SIM options –help-bearer Show bearer options –help-sms Show SMS options

Application Options: -v, –verbose Run action with verbose logs -V, –version Print version -a, –async Use asynchronous methods –timeout=[SECONDS] Timeout for the operation

#### mmeowlink-send.py

##### Help

usage: mmeowlink-send.py [-h] [–serial SERIAL] [–port PORT] [–no-op] [–skip-prelude] [–no-rf-prelude] [–skip-postlude] [-v] [–rf-minutes SESSION_LIFE] [–auto-init] [–init] [–radio_type {mmcommander,subg_rfspy}] [–mmcommander] [–subg_rfspy] [–prefix-path PREFIX_PATH] [–saveall] [–prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc
[–postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,Guardia
{sleep,tweak,ManualCommand} ...

mmeowlink adapter to decocare's SendMsgApp

positional arguments: {sleep,tweak,ManualCommand} Main thing to do between –prefix and–postfix sleep Just sleep between command sets tweak Tweak a builtin command ManualCommand Customize a command

optional arguments: -h, –help show this help message and exit –serial SERIAL serial number of pump [default: ] –port PORT Path to device [default: scan] –no-op Dry run, don't do main function –skip-prelude Don't do the normal prelude. –no-rf-prelude Do the prelude, but don't query the pump. –skip-postlude Don't do the normal postlude. -v, –verbose Verbosity –rf-minutes SESSION_LIFE How long RF sessions should last –auto-init Send power ctrl to initialize RF session. –init Send power ctrl to initialize RF session. –radio_type {mmcommander,subg_rfspy} –mmcommander –subg_rfspy –prefix-path PREFIX_PATH Prefix to store saved files when using –save or –saveall. –saveall Whether or not to save all responses. –prefix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc
Built-in commands to run before the main one. –postfix {BaseCommand,Bolus,Experiment_OP161,Experiment_OP162,FilterGlucoseHistory,FilterHistory,FilterISIGHistory,GuardianSensorAlarmSilenc
Built-in commands to run after the main one.

#### mmtune.py

##### Help

Usage: mmtune.py /dev/ttyMFD1 pumpserial [radio_locale] Radio locale defaults to 'US'. Set to 'WW' for other countries

For command line tools.

## 5.2 API

### 5.2.1 API Reference

#### openaps

#### openaps

**openaps package**

**Subpackages**

**openaps.alias package**

**Submodules**

**openaps.alias.add module** add - add an alias

`openaps.alias.add.`**`configure_app`**(*app*, *parser*)

`openaps.alias.add.`**`main`**(*args*, *app*)

**openaps.alias.alias module**

**class** `openaps.alias.alias.`**`Alias`**(*name=None*, *command=None*, ***kwds*)

Bases: *openaps.configurable.Configurable*

**classmethod `FromConfig`**(*klass*, *config*)

**`fields` = {}**

**`name` = None**

**`optional` = []**

**`prefix` = 'alias'**

**`remove`**(*config*)

**`required` = ['command']**

**`section_name`**()

**`store`**(*config*)

**`url_template` = '{name:s} {command:s}'**

**openaps.alias.remove module** remove - remove an alias

`openaps.alias.remove.`**`main`**(*args*, *app*)

**openaps.alias.show module** show - show all aliases

`openaps.alias.show.`**`configure_app`**(*app*, *parser*)

`openaps.alias.show.`**`main`**(*args*, *app*)

**Module contents**

**class** openaps.alias.**AliasAction**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

**setup_application**()

**class** openaps.alias.**AliasManagement**(*parent*)

Bases: *openaps.cli.commandmapapp.CommandMapApp*

aliases - manage alias configurations

**Subcommand**

alias of *AliasAction*

**get_commands**()

**get_dest**()

**name = 'action'**

**title = '## Alias Menu'**

**class** openaps.alias.**Exported**

Bases: object

**Command**

alias of *AliasManagement*

**Configurable**

alias of Alias

**Subcommand**

alias of *AliasAction*

**classmethod get_configurables**(*Klass*, *conf*)

**classmethod get_map**(*Klass*, *conf*)

**classmethod get_names**(*Klass*, *conf*)

openaps.alias.**get_alias_map**(*conf*)

### openaps.cli package

#### Submodules

#### openaps.cli.commandmapapp module

**class** openaps.cli.commandmapapp.**CommandMapApp**(*parent*)

Bases: object

**class Subcommand**(*method=None*, *parent=None*)

Bases: object

**configure_parser**(*parser*)

**configure_subparser**(*subparser*)

**formatter_class**

alias of RawDescriptionHelpFormatter

**get_description**()

**get_epilog**()

**get_help**()

**setup_application**()
Allows us to use method, injected as dependency earlier to set up argparser before autocompletion/running the app.

CommandMapApp.**commands** = {}

CommandMapApp.**configure_commands**(*parser*)

CommandMapApp.**get**(*name*)

CommandMapApp.**get_commands**()

CommandMapApp.**get_description**()

CommandMapApp.**get_dest**()

CommandMapApp.**get_help**()

CommandMapApp.**get_metavar**()

CommandMapApp.**get_title**()

CommandMapApp.**makeSubcommand**(*ctx*)

CommandMapApp.**metavar** = None

CommandMapApp.**selected**(*args*)

**openaps.cli.helpers module**

openaps.cli.helpers.**format_ini**(*report*)

openaps.cli.helpers.**format_json**(*report*)

openaps.cli.helpers.**format_url**(*report*)

openaps.cli.helpers.**install_show_arguments**(*parser*)

**openaps.cli.subcommand module**

**class** openaps.cli.subcommand.**Subcommand**(*method=None*, *parent=None*)
Bases: object

**configure_parser**(*parser*)

**configure_subparser**(*subparser*)

**formatter_class**
alias of RawDescriptionHelpFormatter

**get_description**()

**get_epilog**()

**get_help**()

**setup_application**()
Allows us to use method, injected as dependency earlier to set up argparser before autocompletion/running the app.

**Module contents**

**class** openaps.cli.**Base**(*args*)

Bases: object

**always_complete_options = True**

**configure_parser**(*parser*)

**epilog**()

**get_described_parser**()

**prep_parser**()

**prolog**()

**run**(*args*)

**class** openaps.cli.**ConfigApp**(*args*)

Bases: *openaps.cli.Base*

**create_git_commit**()

**epilog**()

**git_repo**()

**prolog**()

**read_config**()

**openaps.devices package**

**Submodules**

**openaps.devices.add module** add - add a new device configuration

openaps.devices.add.**configure_app**(*app*, *parser*)

openaps.devices.add.**configure_parser**(*parser*)

openaps.devices.add.**main**(*args*, *app*)

**openaps.devices.device module**

**class** openaps.devices.device.**Device**(*name*, *vendor*)

Bases: *openaps.configurable.Configurable*

**classmethod FromConfig**(*klass*, *vendors*, *config*)

**classmethod FromImport**(*klass*, *candidate*, *config=None*)

**format_url**()

**get**(*k*, **args*)

**optional = []**

**prefix = 'device'**

**read**(*args=None*, *config=None*)

**register_uses**(*uses*)

**required = ['name', 'vendor']**

**store**(*config*)

**vendor = None**

**class** openaps.devices.device.**ExtraConfig**(*name*, ***kwargs*)

Bases: *openaps.configurable.Configurable*

**prefix = 'device'**

**openaps.devices.remove module** remove - remove a device configuration

openaps.devices.remove.**main**(*args*, *app*)

**openaps.devices.show module** show - show all devices

openaps.devices.show.**configure_app**(*app*, *parser*)

openaps.devices.show.**main**(*args*, *app*)

**Module contents**

**class** openaps.devices.**DeviceConfig**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

**setup_application**()

**class** openaps.devices.**Exported**

Bases: object

**Command**

alias of *DeviceConfig*

**Configurable**

alias of Device

**classmethod get_configurables**(*Klass*, *conf*)

**classmethod get_map**(*Klass*, *conf*)

**classmethod get_names**(*Klass*, *conf*)

openaps.devices.**configure_commands**(*parser*, *parent=None*)

openaps.devices.**get_device_map**(*conf*)

openaps.devices.**get_device_names**(*conf*)

openaps.devices.**get_devices**(*conf*)

openaps.devices.**setup_application**(*app*, *parser*)

**openaps.glucose package**

**Submodules**

**openaps.glucose.convert module**

**class** openaps.glucose.convert.**Convert**

Bases: object

How to convert from mg/dL (World Wide format) to mmol/L (mostly UK and ex-UK colonies)

Please note that rounding of these values is a *view* related function, and should happen at the very last point before data is being viewed, not here. See http://physics.stackexchange.com/a/63330

This code *could* be used for mathematical processing of results by someone down the line, so we take pain to avoid throwing away potentially significant data by rounding.

**MMOLL_CONVERT_FACTOR = 18.0**

**classmethod mg_dl_to_mmol_l** (*klass*, *mg_dl*)

**classmethod mmol_l_to_mg_dl** (*klass*, *mmol_l*)

**openaps.glucose.display module**

**class** openaps.glucose.display.**Display**

Bases: object

Round Glucose values for display, so that they are consistent in all OpenAPS apps

Example:

from openaps.glucose.display import Display print(Display.display('mmol/L', 5.5)) print(Display.display('mg/dL', 100))

**classmethod display** (*klass*, *unit*, *val*)

**Module contents**

**openaps.reports package**

**Subpackages**

**openaps.reports.reporters package**

**Submodules**

**openaps.reports.reporters.JSON module**

openaps.reports.reporters.JSON.**date_handler**(*obj*)

openaps.reports.reporters.JSON.**serialize**(*blob*, *reporter*)

**openaps.reports.reporters.base module**

openaps.reports.reporters.base.**serialize**(*blob*, *reporter*)

**openaps.reports.reporters.stdout module**

openaps.reports.reporters.stdout.**close_output_stream**(*reporter*)

openaps.reports.reporters.stdout.**get_output_stream**(*reporter*)

openaps.reports.reporters.stdout.**serialize**(*blob*, *reporter*)

**openaps.reports.reporters.text module**

openaps.reports.reporters.text.**serialize**(*blob*, *reporter*)

**Module contents**

**class** openaps.reports.reporters.**Reporter**(*report*, *device*, *task*)

Bases: object

**close**()

**no_op_serialize**(*data*)

**serialize**(*data*)

openaps.reports.reporters.**default_close_stream**(*reporter*)

openaps.reports.reporters.**default_prep_stream**(*reporter*)

openaps.reports.reporters.**get_reporter_map**()

openaps.reports.reporters.**get_reporters**()

**Submodules**

**openaps.reports.add module** add - add a new report configuration

openaps.reports.add.**configure_app**(*app*, *parser*)

openaps.reports.add.**configure_parser**(*parser*)

openaps.reports.add.**main**(*args*, *app*)

**openaps.reports.invoke module** invoke - generate a report

openaps.reports.invoke.**configure_app**(*app*, *parser*)

openaps.reports.invoke.**main**(*args*, *app*)

**openaps.reports.remove module** remove - remove a device configuration

openaps.reports.remove.**main**(*args*, *app*)

**openaps.reports.report module**

**class** openaps.reports.report.**Report**(*name=None*, *report=None*, *reporter=None*, *device=None*, *use=None*, ***kwds*)

Bases: *openaps.configurable.Configurable*

**fields = {}**

**name = None**

**optional = []**

**prefix = 'report'**

**required = ['report', 'reporter', 'device', 'use']**

**url_template = '{device:s}://{reporter:s}/{use:s}/{name:s}'**

**openaps.reports.show module** show - show all reports

**class** openaps.reports.show.**Formatter**(*app*)
Bases: object

**format_cli**(*report*)

openaps.reports.show.**configure_app**(*app*, *parser*)

openaps.reports.show.**main**(*args*, *app*)

**Module contents**

**class** openaps.reports.**Exported**
Bases: object

**Command**
alias of *ReportManagementActions*

**Configurable**
alias of Report

**Subcommand**
alias of *ReportAction*

**classmethod get_configurables**(*Klass*, *conf*)

**classmethod get_map**(*Klass*, *conf*)

**classmethod get_names**(*Klass*, *conf*)

**class** openaps.reports.**ReportAction**(*method=None*, *parent=None*)
Bases: *openaps.cli.subcommand.Subcommand*

**setup_application**()

**class** openaps.reports.**ReportManagementActions**(*parent*)
Bases: *openaps.cli.commandmapapp.CommandMapApp*

reports - manage report configurations

**Subcommand**
alias of *ReportAction*

**get_commands**()

**get_dest**()

**name = 'action'**

**title = '## Reports Menu'**

openaps.reports.**get_devices**(*conf*)

openaps.reports.**get_report_map**(*conf*)

openaps.reports.**get_report_names**(*conf*)

**openaps.uses package**

**Submodules**

**openaps.uses.registry module**

**class** openaps.uses.registry.**Registry**

Bases: object

**get_uses**(*device*, *config*)

**openaps.uses.use module** use - module for openaps devices to re-use

**class** openaps.uses.use.**Use**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

A no-op base use. A Use is a mini well-behaved linux application. openaps framework will initialize your Use with a *method* and *parent* objects, which are contextual objects within openaps.

**after_main**(*args*, *app*)

**before_main**(*args*, *app*)

**from_ini**(*fields*)

**get_params**(*args*)

Return dictionary of all parameters collected from args.

**main**(*args*, *app*)

Put main app logic here. print "HAHA", args, app

**to_ini**(*args*)

**Module contents**

**class** openaps.uses.**DeviceUsageMap**(*device=None*, *parent=None*)

Bases: *openaps.cli.commandmapapp.CommandMapApp*

Map of uses for specific device

**Subcommand**

alias of *DeviceUsageTask*

**get_commands**()

**get_description**()

**get_dest**()

**get_help**()

**get_metavar**()

**get_title**()

**class** openaps.uses.**DeviceUsageTask**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

One use

**setup_application**()

**class** openaps.uses.**UseDeviceCommands**(*devices=None*, *parent=None*, *config=None*)

Bases: *openaps.cli.commandmapapp.CommandMapApp*

device - which device to use

**Subcommand**

alias of *UseDeviceTask*

**get_commands**()

**get_description**()

**get_dest**()

**get_help**()

**get_title**()

**metavar = 'device'**

**class** openaps.uses.**UseDeviceTask**(*method=None*, *parent=None*)
Bases: *openaps.cli.subcommand.Subcommand*

Manage device usage

Per vendor usage tasks for a device.

**get_description**()

**get_help**()

**setup_application**()

openaps.uses.**all_uses**(*config*, *device*)

openaps.uses.**get_uses_for**(*device*, *parent=None*)

openaps.uses.**known_uses**(*config*, *device*)

openaps.uses.**no_uses**(*device*, *config*)

openaps.uses.**plugin_uses**(*config*, *device*)

### openaps.vendors package

#### Subpackages

### openaps.vendors.plugins package

#### Submodules

**openaps.vendors.plugins.add module** Add a new vendor plugin to openaps-environment.

openaps.vendors.plugins.add.**configure_app**(*app*, *parser*)

openaps.vendors.plugins.add.**main**(*args*, *app*)

**openaps.vendors.plugins.remove module** Remove vendor plugin from openaps-environment

openaps.vendors.plugins.remove.**main**(*args*, *app*)

**openaps.vendors.plugins.show module** Show/list vendor plugins

openaps.vendors.plugins.show.**configure_app**(*app*, *parser*)

openaps.vendors.plugins.show.**main**(*args*, *app*)

**openaps.vendors.plugins.vendor module**

**class** openaps.vendors.plugins.vendor.**Vendor**(*name=None*, ***kwds*)

Bases: *openaps.configurable.Configurable*

**fields = {}**

**get_module**()

**name = None**

**optional = []**

**prefix = 'vendor'**

**required = ['name']**

**url_template = '{name:s}://'**

**Module contents**

**class** openaps.vendors.plugins.**Exported**

Bases: object

**Command**

alias of *VendorManagementActions*

**Configurable**

alias of Vendor

**Subcommand**

alias of *VendorAction*

**classmethod get_configurables**(*Klass*, *conf*)

**classmethod get_map**(*Klass*, *conf*)

**classmethod get_names**(*Klass*, *conf*)

**class** openaps.vendors.plugins.**VendorAction**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

**setup_application**()

**class** openaps.vendors.plugins.**VendorManagementActions**(*parent*)

Bases: *openaps.cli.commandmapapp.CommandMapApp*

vendors - manage vendor plugin configurations

**Subcommand**

alias of *VendorAction*

**get_commands**()

**get_dest**()

**name = 'command'**

**title = '## Vendors Menu'**

openaps.vendors.plugins.**get_plugins**(*conf*)

openaps.vendors.plugins.**get_vendor_map**(*conf*)

openaps.vendors.plugins.**get_vendor_names**(*conf*)

**Submodules**

**openaps.vendors.dexcom module** Dexcom - openaps driver for dexcom

**class** openaps.vendors.dexcom.**DescribeClocks**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

Describe all the clocks

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**EGVRecord**
Bases: openaps.vendors.dexcom.EGV

**to_dict**()

**class** openaps.vendors.dexcom.**GetFirmwareHeader**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**ReadBatteryLevel**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**ReadBatteryState**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**ReadBlindedMode**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Blinded Mode

**class** openaps.vendors.dexcom.**ReadChargerCurrentSetting**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Charger current setting

**class** openaps.vendors.dexcom.**ReadClockMode**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Clock Mode

**class** openaps.vendors.dexcom.**ReadDeviceMode**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Device Mode

**class** openaps.vendors.dexcom.**ReadDisplayTime**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Display Time Offset

**class** openaps.vendors.dexcom.**ReadDisplayTimeOffset**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.ReadSystemTimeOffset*

Read Display Time Offset

**class** openaps.vendors.dexcom.**ReadGlucoseUnit**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Glucose Unit

**class** openaps.vendors.dexcom.**ReadHardwareBoardId**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Hardware board ID

**class** openaps.vendors.dexcom.**ReadLanguage**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Language

**class** openaps.vendors.dexcom.**ReadManufacturingData**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**ReadRTC**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read RTC

**class** openaps.vendors.dexcom.**ReadSetupWizardState**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read Setup wizard state

**class** openaps.vendors.dexcom.**ReadSystemTime**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read System Time

**class** openaps.vendors.dexcom.**ReadSystemTimeOffset**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.SameNameCommand*

Read System Time Offset

**pass_result**(*result*)

**class** openaps.vendors.dexcom.**ReadTransmitterId**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**SameNameCommand**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**pass_result**(*result*)

**class** openaps.vendors.dexcom.**SensorRecord**
Bases: openaps.vendors.dexcom.Sensor

**to_dict**()

**class** openaps.vendors.dexcom.**UpdateTime**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

Update receiver time

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**get_program**(*args*)

**main**(*args*, *app*)

**upload_program**(*program*)

**class** openaps.vendors.dexcom.**WriteChargerCurrentSetting**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**MAP = ['Off', 'Power100mA', 'Power500mA', 'PowerMax', 'PowerSuspended']**

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**battery**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**calibrations**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

read calibration entry records

**main**(*args*, *app*)
Implement a main method that takes args and app as parameters. Use self.dexcom.Read... to get data. Return the resulting data for this task/command. The data will be passed to prerender_<format> by the reporting system.

**prerender_json**(*data*)
since everything is a dict/strings/ints, we can pass thru to json

**prerender_stdout**(*data*)

**prerender_text**(*data*)
turn everything into a string

**class** openaps.vendors.dexcom.**config**(*method=None*, *parent=None*)
Bases: *openaps.uses.use.Use*

**configure_app**(*app*, *parser*)

**main**(*args*, *app*)

openaps.vendors.dexcom.**display_device**(*device*)

openaps.vendors.dexcom.**fix_display_time**(*display_time=None*, ***kwds*)

**class** openaps.vendors.dexcom.**glucose**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

This is a good example of what is needed for new commands. To add additional commands, subclass from scan as shown.

**RECORD_TYPE = 'EGV_DATA'**

**TEXT_COLUMNS = ['display_time', 'glucose', 'trend_arrow']**

**main**(*args*, *app*)
Implement a main method that takes args and app as parameters. Use self.dexcom.Read... to get data. Return the resulting data for this task/command. The data will be passed to prerender_<format> by the reporting system.

**prerender_json**(*data*)
since everything is a dict/strings/ints, we can pass thru to json

**prerender_stdout**(*data*)

**prerender_text**(*data*)
turn everything into a string

**class** openaps.vendors.dexcom.**insertion_time**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.sensor*

Fetch INSERTION_TIME records from the Dexcom receiver.

These are created when sensors are started.

**RECORD_TYPE = 'INSERTION_TIME'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'insertion_time', 'session_state']**

**class** openaps.vendors.dexcom.**iter_calibrations**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.calibrations*

read last <count> calibration records, default 10, eg:

- iter_calibrations - read last 10 calibration records
- iter_calibrations 2 - read last 2 calibration records

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**iter_calibrations_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.calibrations*

read last <hours> of calibration records, default 1, eg:

- iter_calibrations_hours - read last 1 hour of calibration records
- iter_calibrations_hours 4.3 - read last 4.3 hours of calibration records

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**iter_glucose**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.glucose*

read last <count> glucose records, default 100, eg:

- iter_glucose - read last 100 records
- iter_glucose 2 - read last 2 records

**RECORD_TYPE = 'EGV_DATA'**

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**iter_glucose_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.glucose*

read last <hours> of glucose records, default 1, eg:

- iter_glucose_hours - read last 1 hour of glucose records
- iter_glucose_hours 4.3 - read last 4.3 hours of glucose records

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**iter_insertion_time**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor*, *openaps.vendors.dexcom.insertion_time*

**RECORD_TYPE = 'INSERTION_TIME'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'insertion_time', 'session_state']**

**class** openaps.vendors.dexcom.**iter_insertion_time_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor_hours*, *openaps.vendors.dexcom.iter_insertion_tim*

**RECORD_TYPE = 'INSERTION_TIME'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'insertion_time', 'session_state']**

**class** openaps.vendors.dexcom.**iter_meter_data**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor*, *openaps.vendors.dexcom.meter_data*

**RECORD_TYPE = 'METER_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'meter_time', 'meter_glucose']**

**class** openaps.vendors.dexcom.**iter_meter_data_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor_hours*, *openaps.vendors.dexcom.iter_meter_data*

**RECORD_TYPE = 'METER_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'meter_time', 'meter_glucose']**

**class** openaps.vendors.dexcom.**iter_sensor**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_glucose*, *openaps.vendors.dexcom.sensor*

**RECORD_TYPE = 'SENSOR_DATA'**

**TEXT_COLUMNS = ['display_time', 'unfiltered', 'filtered', 'rssi']**

**class** openaps.vendors.dexcom.**iter_sensor_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_glucose_hours*, *openaps.vendors.dexcom.iter_sensor*

**RECORD_TYPE = 'SENSOR_DATA'**

**TEXT_COLUMNS = ['display_time', 'unfiltered', 'filtered', 'rssi']**

**class** openaps.vendors.dexcom.**iter_sensor_insertions**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.sensor_insertions*

read last <count> sensor insertion, removal, and expiration records, default 10, eg:

- iter_sensor_insertions - read last 10 records
- iter_sensor_insertions 2 - read last 2 records

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.dexcom.**iter_sensor_insertions_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.sensor_insertions*

read last <hours> of sensor insertion, removal, and expiration records, default 1, eg:

•iter_sensor_insertions_hours - read last 1 hour of sensor insertion, removal, and expiration records

•iter_sensor_insertions_hours 4.3 - read last 4.3 hours of sensor insertion, removal, and expiration records

**configure_app** (*app*, *parser*)

**get_params** (*args*)

**main** (*args*, *app*)

**class** openaps.vendors.dexcom.**iter_user_event_data** (*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor*, *openaps.vendors.dexcom.user_event_data*

**RECORD_TYPE = 'USER_EVENT_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'event_type', 'event_sub_type', 'event_value']**

**class** openaps.vendors.dexcom.**iter_user_event_data_hours** (*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.iter_sensor_hours*, *openaps.vendors.dexcom.iter_user_event_da*

**RECORD_TYPE = 'USER_EVENT_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'event_type', 'event_sub_type', 'event_value']**

**class** openaps.vendors.dexcom.**meter_data** (*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.sensor*

Fetch METER_DATA records from the Dexcom receiver.

**RECORD_TYPE = 'METER_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'meter_time', 'meter_glucose']**

**class** openaps.vendors.dexcom.**oref0_glucose** (*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.glucose*

Get Dexcom glucose formatted for Nightscout, merged with raw data. [#oref0]

**NS_DIRECTIONS = {'DoubleUp': 1, 'Flat': 4, 'SingleDown': 6, 'SingleUp': 2, 'DoubleDown': 7, None: 0, 'FortyFiveDow**

**NS_NAMES = [None, 'DoubleUp', 'SingleUp', 'FortyFiveUp', 'Flat', 'FortyFiveDown', 'SingleDown', 'DoubleDown', 'NO**

**TEXT_COLUMNS = ['display_time', 'glucose', 'trend_arrow']**

**adjust_dates** (*item*)

**static arrow_to_trend** (*arrow*)

**configure_app** (*app*, *parser*)

**from_ini** (*fields*)

**get_glucose_data** (*params*, *args*)

**get_params** (*args*)

**get_sensor_data** (*params*, *args*)

**main** (*args*, *app*)

**to_ini** (*args*)

**classmethod trend_to_direction** (*Klass*, *trend*, *arrow*)

openaps.vendors.dexcom.**parse_clock** (*candidate*)

**class** openaps.vendors.dexcom.**scan**(*method=None*, *parent=None*)
Bases: *openaps.uses.use.Use*

scan for usb stick

**before_main**(*args*, *app*)

**main**(*args*, *app*)

**scanner**()

**class** openaps.vendors.dexcom.**sensor**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.glucose*

Fetch Sensor (raw) records from Dexcom receiver.

Fetches raw records.

**RECORD_TYPE = 'SENSOR_DATA'**

**TEXT_COLUMNS = ['display_time', 'unfiltered', 'filtered', 'rssi']**

**class** openaps.vendors.dexcom.**sensor_insertions**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.scan*

read sensor insertion, removal, and expiration records of sensors

**main**(*args*, *app*)
Implement a main method that takes args and app as parameters. Use self.dexcom.Read... to get data. Return the resulting data for this task/command. The data will be passed to prerender_<format> by the reporting system.

**prerender_json**(*data*)
since everything is a dict/strings/ints, we can pass thru to json

**prerender_stdout**(*data*)

**prerender_text**(*data*)
turn everything into a string

openaps.vendors.dexcom.**set_config**(*args*, *device*)

**class** openaps.vendors.dexcom.**user_event_data**(*method=None*, *parent=None*)
Bases: *openaps.vendors.dexcom.sensor*

Fetch USER_EVENT_DATA records from the Dexcom receiver.

**RECORD_TYPE = 'USER_EVENT_DATA'**

**TEXT_COLUMNS = ['display_time', 'system_time', 'event_type', 'event_sub_type', 'event_value']**

**openaps.vendors.medtronic module** Medtronic - openaps driver for Medtronic

**class** openaps.vendors.medtronic.**InputProgramRequired**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**get_program**(*args*)

**main**(*args*, *app*)

**upload_program**(*program*)

**class** openaps.vendors.medtronic.**MedtronicTask**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.scan*

**MAX_SESSION_DURATION = 10**

**after_main**(*args*, *app*)

**before_main**(*args*, *app*)

**check_session**(*app*)

**create_session**()

**get_model**()

**get_session_info**()

**main**(*args*, *app*)

**read_session_file**()

**record_stats = True**

**requires_session = True**

**save_session = True**

**setup_medtronic**()

**update_session_info**(*fields*)

**write_session_file**(*session*)

**class** openaps.vendors.medtronic.**SameNameCommand**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**SelectedNameCommand**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**Session**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

session for pump

**configure_parser**(*parser*)

**main**(*args*, *app*)

**requires_session = False**

**setup_application**()

**class** openaps.vendors.medtronic.**bolus**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.InputProgramRequired*

Send bolus command. [#warning!!!]

Beware! This is a powerful command because it can give a lot of insulin. Please be careful! Not a part of oref0.

**Requires json input with the following keys defined:**

- *units* - Number of units to bolus.

Zero point one units: { “units”: 0.1 }

Two units: { “units”: 2 }

**upload_program**(*program*)

**class** openaps.vendors.medtronic.**config**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

**before_main**(*args*, *app*)

**configure_app**(*app*, *parser*)

**main**(*args*, *app*)

**requires_session = False**

openaps.vendors.medtronic.**configure_add_app**(*app*, *parser*)

openaps.vendors.medtronic.**configure_app**(*app*, *parser*)

openaps.vendors.medtronic.**configure_parser**(*parser*)

openaps.vendors.medtronic.**configure_use_app**(*app*, *parser*)

openaps.vendors.medtronic.**display_device**(*device*)

**class** openaps.vendors.medtronic.**filter_glucose_date**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Search for glucose pages including begin and end dates (iso 8601).

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**class** openaps.vendors.medtronic.**filter_isig_date**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.filter_glucose_date*

Search for isig pages including begin and end dates (iso 8601).

openaps.vendors.medtronic.**get_uses**(*device*, *config*)

**class** openaps.vendors.medtronic.**iter_glucose**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Read latest 100 glucose records

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**maxCount = 99**

**range**()

**class** openaps.vendors.medtronic.**iter_glucose_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Read latest n hours of glucose data

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**get_record_timestamp**(*record*)

**main**(*args*, *app*)

**range**()

**class** openaps.vendors.medtronic.**iter_pump**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.iter_glucose*

Read latest 100 pump records

**range**()

**class** openaps.vendors.medtronic.**iter_pump_hours**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.iter_glucose_hours*

Read latest n hours of pump records

**get_record_timestamp**(*record*)

**range**()

**class** openaps.vendors.medtronic.**key_presser**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

**run_presses**(*recipe*)

openaps.vendors.medtronic.**main**(*args*, *app*)
print "MEDTRONIC", args, app print "app commands", app.selected.name

**class** openaps.vendors.medtronic.**model**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Get model number [#oref0] [#recommended] [#safe]

This is one of the safest commands available.

**configure_app**(*app*, *parser*)

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**mytest**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Testing read_settings

**main**(*args*, *app*)

**requires_session** = **False**

openaps.vendors.medtronic.**parse_clock**(*candidate*)

**class** openaps.vendors.medtronic.**press_keys**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.key_presser*

Press keys

**configure_app**(*app*, *parser*)

**from_ini**(*fields*)

**get_params**(*args*)

**main**(*args*, *app*)

**to_ini**(*args*)

**class** openaps.vendors.medtronic.**read_basal_profile_A**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read basal profile A.

**class** openaps.vendors.medtronic.**read_basal_profile_B**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read basal profile B.

**class** openaps.vendors.medtronic.**read_basal_profile_std**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read default basal profile.

**class** openaps.vendors.medtronic.**read_battery_status**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Check battery status. [#oref0]

**class** openaps.vendors.medtronic.**read_bg_targets**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SelectedNameCommand*

Read bg targets. [#oref0]

**selected = 'read_bg_targets'**

**class** openaps.vendors.medtronic.**read_carb_ratios**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read carb_ratios. [#oref0]

**class** openaps.vendors.medtronic.**read_clock**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Read date/time of pump [#oref0]

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**read_current_glucose_pages**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read current glucose pages.

**class** openaps.vendors.medtronic.**read_current_history_pages**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read current history pages.

**class** openaps.vendors.medtronic.**read_glucose_data**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read pump glucose page

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**class** openaps.vendors.medtronic.**read_history_data**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Read pump history page

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**read_insulin_sensitivies**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

XXX: Deprecated. Don't use. Use read_insulin_sensitivities instead.

**class** openaps.vendors.medtronic.**read_insulin_sensitivities**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read insulin sensitivities. [#oref0]

**class** openaps.vendors.medtronic.**read_selected_basal_profile**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Fetch the currently selected basal profile. [#oref0]

**class** openaps.vendors.medtronic.**read_settings**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read settings. [#oref0]

**class** openaps.vendors.medtronic.**read_status**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Get pump status

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**read_temp_basal**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Read temporary basal rates. [#oref0]

**class** openaps.vendors.medtronic.**reservoir**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Get pump remaining insulin

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**resume_pump**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.suspend_pump*

resume pumping.

**class** openaps.vendors.medtronic.**scan**(*method=None*, *parent=None*)
Bases: *openaps.uses.use.Use*

scan for usb stick

**configure_app**(*app*, *parser*)

**main**(*args*, *app*)

**scanner**()

**class** openaps.vendors.medtronic.**set_clock**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.InputProgramRequired*

Set clock.

input should be the name of a json file, or - for stdin. The json file should contain a key named *clock* with the new requested value formatted as ISO 8601 including seconds.

{ "clock": "2016-03-14T14:23:40" }

Alternatively, the *–to* switch maybe used: The keyword *now* may be given to automatically generate a request using the system's current time. Otherwise, the value must be something that can be parsed as an ISO 8601 formatted time including the seconds.

–to now –to $(date -Iseconds) –to 2016-03-14T14:29:41-0700

**configure_app**(*app*, *parser*)

**get_params**(*args*)

**get_program**(*args*)

**upload_program**(*program*)

openaps.vendors.medtronic.**set_config**(*args*, *device*)

**class** openaps.vendors.medtronic.**set_temp_basal**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.InputProgramRequired*

Set temporary basal rates. [#oref0]

**Requires json input with the following keys defined:**

- *temp* - the type of temporary rate, *percent* or *absolute*
- *rate* - The temporary rate, in units. Examples, 0.0, 1.2, 0.1
- *duration* - The duration in minutes of the temporary rate. The duration must be multiples of 30 minutes.

Eg, actively canceling a rate: { "temp": "absolute", "rate": 0, "duration": 0 }

Zero basal for half hour: { "temp": "absolute", "rate": 0, "duration": 30 }

One and a half units for one hour: { "temp": "absolute", "rate": 1.5, "duration": 60 }

**required_inputs = ['duration', 'rate']**

**upload_program**(*program*)

**class** openaps.vendors.medtronic.**settings**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.MedtronicTask*

Get pump settings

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**status**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.read_status*

Get pump status (alias for read_status)

**class** openaps.vendors.medtronic.**suspend_pump**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.SameNameCommand*

Suspend pumping.

**main**(*args*, *app*)

**class** openaps.vendors.medtronic.**test_oref0_compat_menu**(*method=None*, *parent=None*)
Bases: *openaps.vendors.medtronic.key_presser*

**main**(*args*, *app*)

**recipe = ['DOWN', 'ESC', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', 'DOWN', '**

**openaps.vendors.process module** process - a fake vendor to run arbitrary commands

openaps.vendors.process.**configure_add_app**(*app*, *parser*)

openaps.vendors.process.**display_device**(*device*)

openaps.vendors.process.**set_config**(*args*, *device*)

**class** openaps.vendors.process.**shell**(*method=None*, *parent=None*)
Bases: *openaps.uses.use.Use*

run a process in a subshell

**configure_app**(*app*, *parser*)

**from_ini**(*fields*)

**get_params**(*args*)

**main**(*args*, *app*)

**prerender_json**(*data*)
since everything is a dict/strings/ints, we can pass thru to json

**to_ini**(*args*)

**openaps.vendors.units module** Units - units tool for openaps

**class** openaps.vendors.units.**ConvertInput**(*method=None*, *parent=None*)
Bases: *openaps.uses.use.Use*

**CONVERTERS = {'mg/dL': <bound method type.mmol_l_to_mg_dl of <class 'openaps.glucose.convert.Convert'>>, 'mmol/**

**configure_app**(*app*, *parser*)

**convert**(*program*)

**get_params**(*args*)

**get_program**(*args*)

**main**(*args*, *app*)

**set_converter**(*args*)

**class** openaps.vendors.units.**bg_targets**(*method=None*, *parent=None*)
Bases: *openaps.vendors.units.ConvertInput*

Convert bg_targets json to preferred unit.

**convert**(*bg_targets*)

openaps.vendors.units.**display_device**(*device*)

openaps.vendors.units.**get_uses**(*device*, *config*)

**class** openaps.vendors.units.**insulin_sensitivities**(*method=None*, *parent=None*)
Bases: *openaps.vendors.units.ConvertInput*

Convert read_insulin_sensitivities json to preferred unit.

**convert**(*insulin_sensitivities*)

openaps.vendors.units.**set_config**(*args*, *device*)

**Module contents**

**class** openaps.vendors.**ChangeVendorApp**(*method=None*, *parent=None*)

Bases: *openaps.cli.subcommand.Subcommand*

Allow subcommand to handle setup_application

**class** openaps.vendors.**Exported**

Bases: object

**Command**

alias of *VendorConfigurations*

**Configurable**

alias of Vendor

**Subcommand**

alias of *ChangeVendorApp*

**classmethod get_configurables**(*Klass*, *conf*)

**classmethod get_map**(*Klass*, *conf*)

**class** openaps.vendors.**VendorConfigurations**(*parent*)

Bases: *openaps.cli.commandmapapp.CommandMapApp*

**Subcommand**

alias of *ChangeVendorApp*

**get_commands**()

**get_dest**()

**get_vendor**(*vendor*)

openaps.vendors.**all_vendors**(*config=None*)

openaps.vendors.**find_plugins**(*config*)

openaps.vendors.**get_configurable_devices**(*ctx*)

openaps.vendors.**get_map**(*config=None*)

openaps.vendors.**get_vendors**()

openaps.vendors.**lookup**(*name*, *config=None*)

openaps.vendors.**lookup_dotted**(*name*, *config=None*)

**Submodules**

**openaps.builtins module**

**class** openaps.builtins.**BuiltinApp**(*args*)

Bases: *openaps.cli.ConfigApp*

**name = 'builtin'**

**class** openaps.builtins.**RunnableAlias**(*spec*, *parent*)

Bases: object

openaps.builtins.**dispatch**(*args*, *back*)

openaps.builtins.**get_alias**(*command*, *app*)

openaps.builtins.**get_builtins**()

openaps.builtins.**is_builtin**(*command*)

**openaps.config module**

**class** openaps.config.**Config**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)

Bases: ConfigParser.SafeConfigParser

**OPTCRE = <_sre.SRE_Pattern object>**

**classmethod Read**(*klass*, *name=None*, *defaults=['/etc/openaps/openaps.ini', '/home/docs/.openaps.ini', 'openaps.ini']*)

**add_device**(*device*)

**fmt**()

Write an .ini-format representation of the configuration state.

**ini_name = 'openaps.ini'**

**remove_device**(*device*)

**save**()

**set_ini_path**(*ini_path='openaps.ini'*)

**openaps.configurable module**

**class** openaps.configurable.**Configurable**(*name*, ***kwargs*)

Bases: object

**classmethod FromConfig**(*klass*, *config*)

**classmethod FromImport**(*klass*, *candidate*, *config=None*)

**add_option**(*k*, *v*)

**format_url**()

**get**(*k*, **args*)

**items**()

**name = None**

**optional = []**

**prefix = '{name:s}'**

**remove**(*config*)

**required = []**

**section_name**()

**store**(*config*)

**url_template = ''**

**openaps.init module**

openaps.init.**init**(*args*)

**Module contents** Release notes:

- **0.1.0 - transition to Exported interfaces, utilize pkg_resources** more for advertisements of openaps capabilities across the python environment. Also, rely more on json import/export of configuration. Also, variety of git tweaks.

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

## O

## A



## B



## C

## D

## E



## F



## G

## I



## K



## L



## M

## N

## O



## P

## R

## S

## T



## U

## V



## W