Building your own homebrew Alarm System Using Particle and VictorOps

I'm a Maker. I like to make things. A few months ago, one of our dev's at VictorOps Greg Frank got me interested in Particle.io Photon Microcontrollers. Greg has built several home automation things with Arduino as well as Photon devices, documented here in a previous blog post.

His projects got me interested in replacing my home alarm system with a homebrew version that could utilize VictorOps to alert me when a door or window opens (and tell me which one). My traditional home security system is a pretty dumb DSC Power 832 system that is built to call out to a call center when something happens. Boring! I want to get a VictorOps Alert when windows open and close!

So, I setoff to roll my own. I have a little hardware background and knew basically how to build the parts needed. My house has wiring to each door and window, and each access point has a reed switch that is engaged by a magnet mounted to the door or window. When the door or window is open the magnet disconnects the reed switch breaking the circuit. Pretty simple. Basically, in theory, if you wire that switch to a digital input on the Photon, you could determine if the door or window was open or closed. Cool! Problem is I have 24 instrumented windows and doors in my system, so my design will need a few more chips to read all those inputs using only and few pins on the Photon.

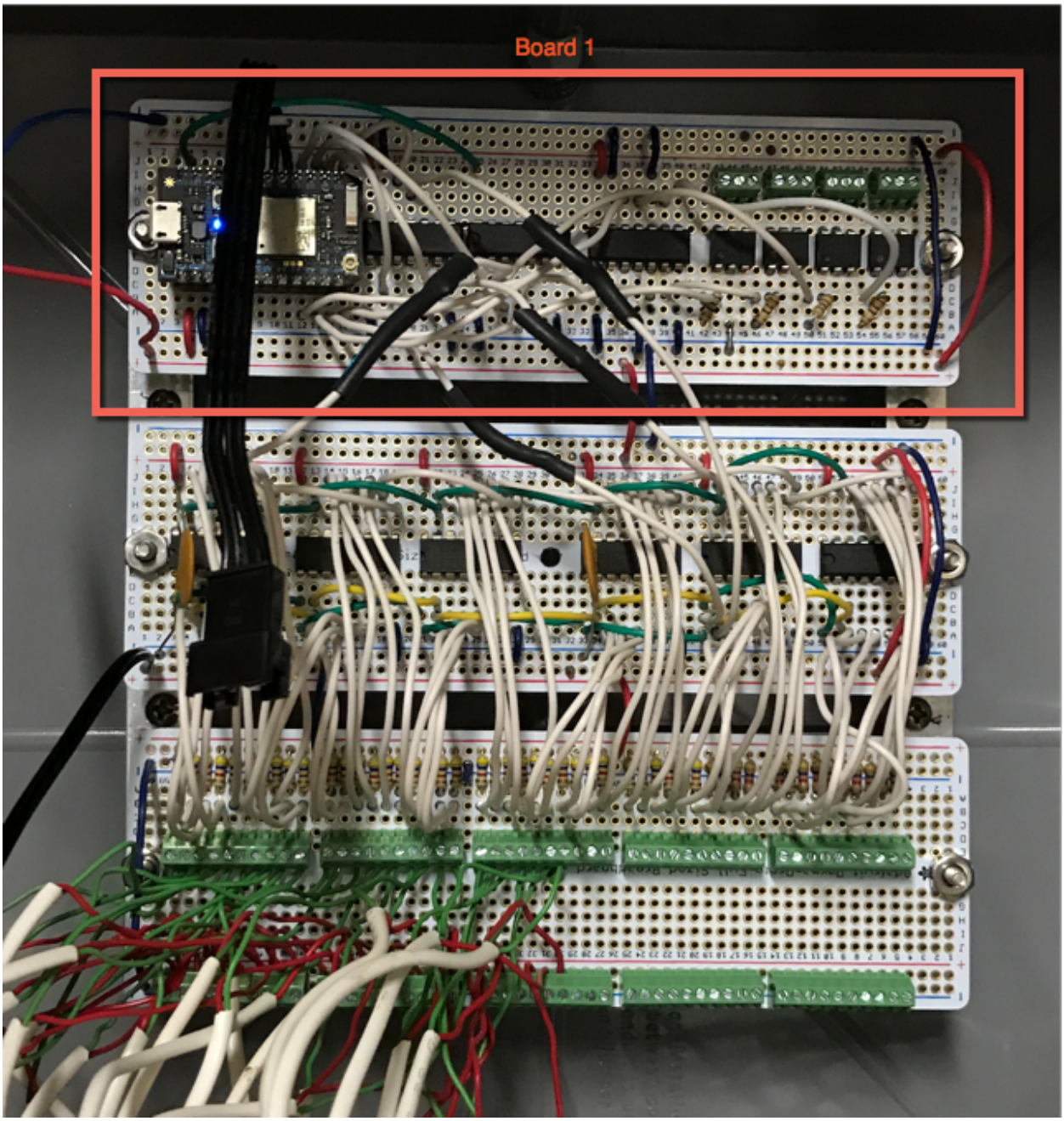
So as I planned my system, my first requirement was that each access point would be its own zone in my system. Most simple commercial systems such as the one in my house basically only have four zones. The installers simply connect all the reed switches in series at the panel for a floor of the house and connect that to one of the four zones. It works but the system doesn't know what window is open, rather what floor it happened on. That was boring, so I decided my system would have 48 zones so each door or window could be individually addressed. My goal was get VictorOps alerts that said something like "Kitchen North Window Opened".

The Particle Photon board is cool. Upon boot up it attaches to your wifi and runs its executive code and any code you have written for it. You can do all your development in cloud on their site with their online IDE. Press the flash button, your code is compiled, uploaded to the Photon, and you're off to the races. Very, very cool.

So I first designed my system on paper and built parts using breadboards to make sure they worked. Then I built more permanent versions where all the components were soldered in.

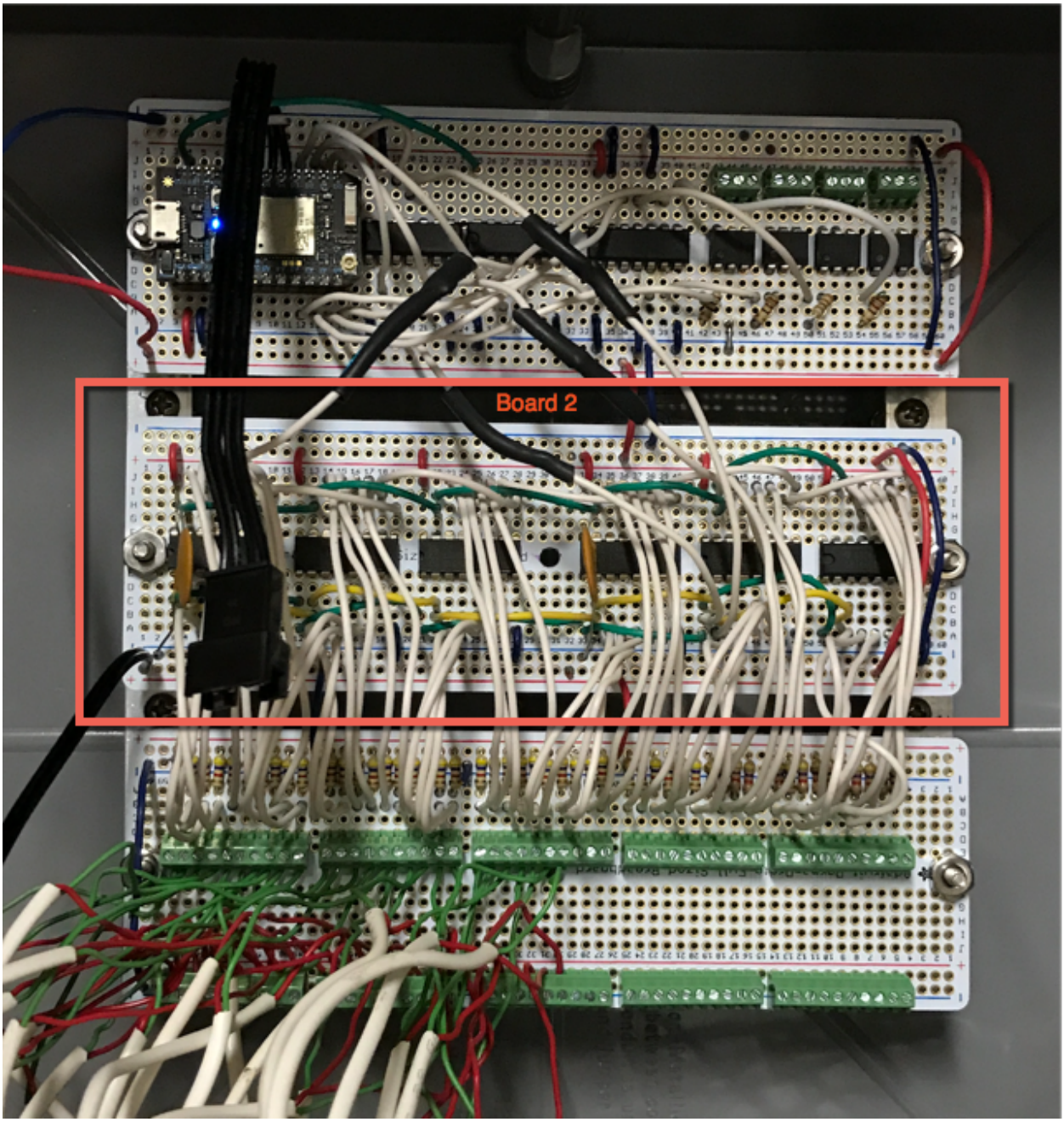
Basically, the system consists of 4 simple boards (only three are required, one is an LED board for display not discussed here). All these boards could easily be on one board but the proto boards I used to make the parts reduce the density of what you can achieve.

Board 1 – Main Logic Board



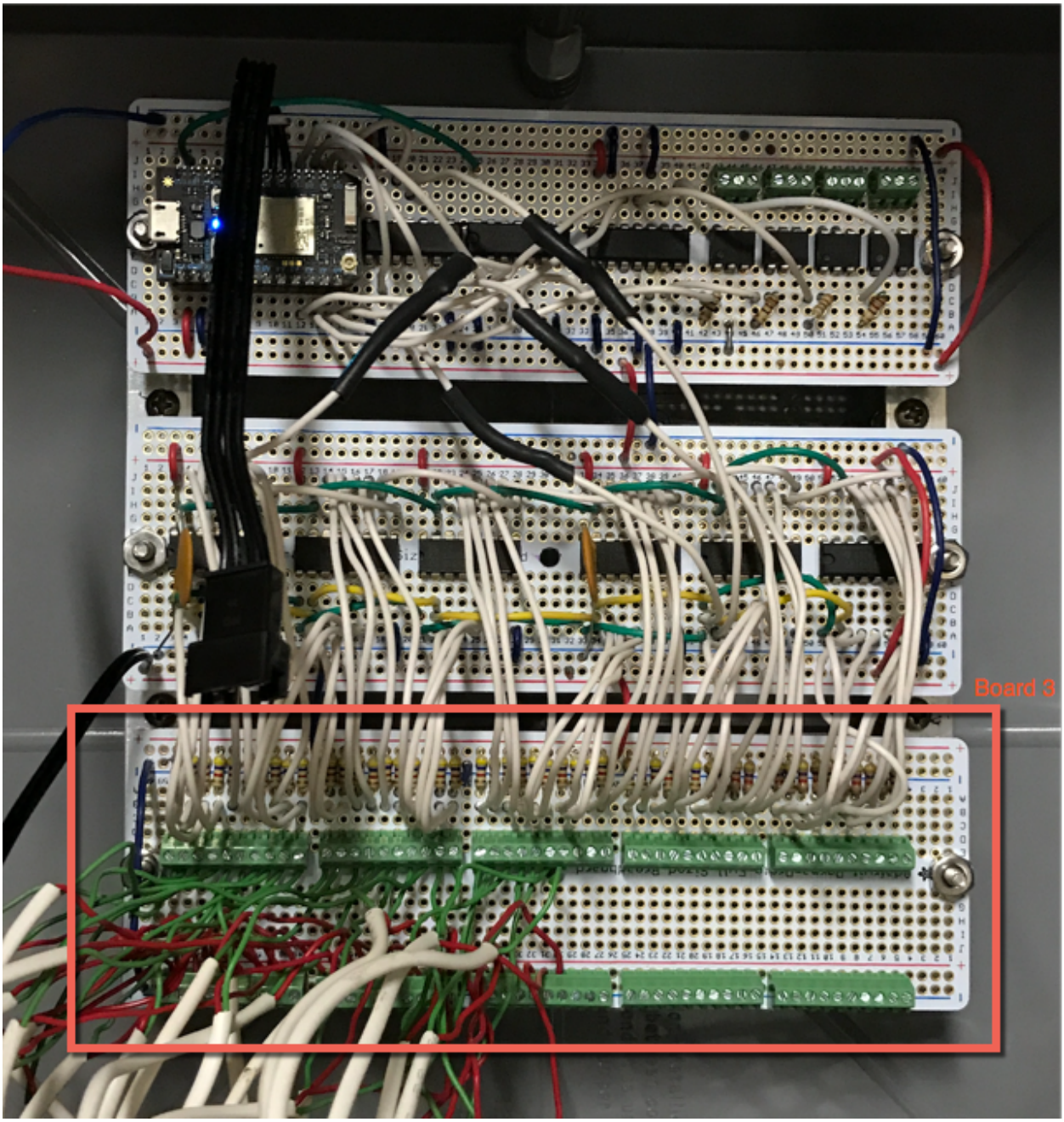
The first board (top board outlined in red) is the Main Logic Board that contains the Photon processor (left) and some supporting chips (middle) that change the logic levels from the Photons 3V range to TTL 5V range. In addition, the four chips on the right are solid state relays that can be used to pass signals through to my legacy alarm system (these are also optional).

Board #2 – Zone input shift registers



The zone input shift register board (center) contains 6 8-bit shift registers wired so they will cascade. Each door or window reed switch will terminate into one bits of one of these 8 bit chips. Basically, the bit state will be a 1 or 0 based on the open or closed state of the window or door. I think you can basically cascade these to your hearts content so you could have infinite inputs if you like.

Board #3: The Terminal board



Finally, the third (lowest) board is simply a terminal strip board with the screw down terminals for the set of wires coming from each door or window. These inputs are jumpered over to the middle board to connect to one of the input pins on a shift register (the white wires). The resistors are pull-up resistors to make sure you get a valid logic level (+5V) when a door or window opens the circuit. Zero volts signifies the window or door is closed in the system. I'm only using about 24 of the zones so far, so I have room for expansion.

The Basic Logic Loop

The Photon executive basically calls a setup() function you define once when it boots up. Then it repeatedly calls your loop() function endlessly. For this project I have the executive delay() for a quarter second between calls so the system basically runs at 4hz.

```
//=============================================================================================
//=============================================================================================
// loop through each of the 48 zones in the system, and update the zone list
//
//=============================================================================================
//=============================================================================================

// load the 48 bits into the cascade input shift register. The 48 bits represent each window or door
// in the house that have reed switches on them and home runned to the location of the security system

zoneInputShiftRegister.load_latch();

// loop through reading each bit (zone) and updationg or zone list of objects to their new state
// the zone will set a flag in itself to signifiy it has changed state so we can use that in later
// processing to send messages and light led's

for (int c=0; c<zoneList.entries(); c++) {

    // read the current output bit representing the window or door
    // (window open == 5 volts, window closed == 0 volts)

    int pinValue = zoneInputShiftRegister.readBit( );

    // get the next zone

    CxZone *zone = zoneList.at( c );

    if ( zone->configured()) {

        // the zone is configured ( used by the system), so set the new state in the object

        if (pinValue == 1) {
            zone->setZoneActivated( TRUE );
        } else {
            zone->setZoneActivated( FALSE );
        }

    } else {

        // the zone is not configured (not used in the current system) so just set it to
        // closed state (even though its actually electically open).  These keeps you
        // from having to jumper unused zones on the input block.

        zone->setZoneActivated( FALSE );
    }

    // shift in the next bit representing the next zone

    zoneInputShiftRegister.shift();

}
```

My basic loop() function reads all 48 zone bits and updates the link list of zones with their new state.

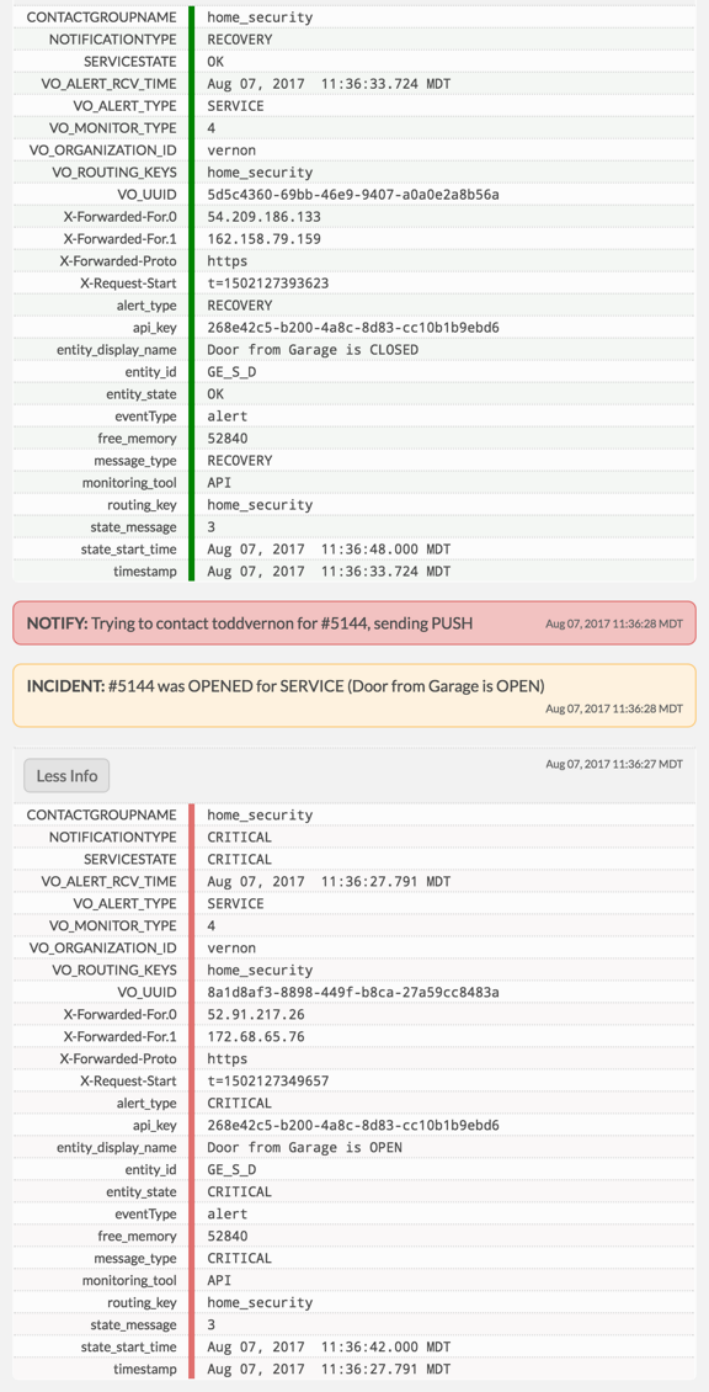
```
//=============================================================================================
//=============================================================================================
// loop through the updated zone list now that we have read in new data.  Each time we see a zone
// that is both configured (used in the ssystem) and changed meaning its state has changed from open
// to closed, or closed to open, we send a change event to the partice cloud with the new state
//
//=============================================================================================
//=============================================================================================
for (int c=0; c<zoneList.entries(); c++) {

    // get the next zone

    CxZone *zone = zoneList.at( c );

    if ( zone->configured() ) {

        if (zone->changed() ) {

            // send zone state change to particle cloud

            CxString json = zone->format_victorops_json();
            Particle.publish( "access_changed" , json.data());
        }
    }
}
```

A second loop then executes and when it finds a zone that has changed (a window has opened or closed) it then builds a particle message to send off to the particle cloud. The message is called access_changed. The json message looks something like this..

```
{
   "channel_number":"SYSTEM",
   "message_type":"CRITICAL",
   "entity_id":"RF_E_W",
   "entity_display_name":"Family Room East Window is OPEN",
   "state_start_time":<seconds from epoch>,
   "free_memory":<amount of free memory in photon>
}
```

I setup a WebHook in the particle cloud to transform this information into a message that is then sent on to the VictorOps endpoint in the same format.

Paging me because the Garage Door is Open!



CONTACTGROUPNAME	home_security
NOTIFICATIONTYPE	RECOVERY
SERVICESTATE	OK
VO_ALERT_RCV_TIME	Aug 07, 2017 11:36:33.724 MDT
VO_ALERT_TYPE	SERVICE
VO_MONITOR_TYPE	4
VO_ORGANIZATION_ID	vernon
VO_ROUTING_KEYS	home_security
VO_UUID	5d5c4360-69bb-46e9-9407-a0a0e2a8b56a
X-Forwarded-For.0	54.209.186.133
X-Forwarded-For.1	162.158.79.159
X-Forwarded-Proto	https
X-Request-Start	t=1502127393623
alert_type	RECOVERY
api_key	268e42c5-b200-4a8c-8d83-cc10b1b9ebd6
entity_display_name	Door from Garage is CLOSED
entity_id	GE_S_D
entity_state	OK
eventType	alert
free_memory	52840
message_type	RECOVERY
monitoring_tool	API
routing_key	home_security
state_message	3
state_start_time	Aug 07, 2017 11:36:48.000 MDT
timestamp	Aug 07, 2017 11:36:33.724 MDT

NOTIFY: Trying to contact toddvernon for #5144, sending PUSH Aug 07, 2017 11:36:28 MDT

INCIDENT: #5144 was OPENED for SERVICE (Door from Garage is OPEN)
Aug 07, 2017 11:36:28 MDT

Less Info Aug 07, 2017 11:36:27 MDT

CONTACTGROUPNAME	home_security
NOTIFICATIONTYPE	CRITICAL
SERVICESTATE	CRITICAL
VO_ALERT_RCV_TIME	Aug 07, 2017 11:36:27.791 MDT
VO_ALERT_TYPE	SERVICE
VO_MONITOR_TYPE	4
VO_ORGANIZATION_ID	vernon
VO_ROUTING_KEYS	home_security
VO_UUID	8a1d8af3-8898-449f-b8ca-27a59cc8483a
X-Forwarded-For.0	52.91.217.26
X-Forwarded-For.1	172.68.65.76
X-Forwarded-Proto	https
X-Request-Start	t=1502127349657
alert_type	CRITICAL
api_key	268e42c5-b200-4a8c-8d83-cc10b1b9ebd6
entity_display_name	Door from Garage is OPEN
entity_id	GE_S_D
entity_state	CRITICAL
eventType	alert
free_memory	52840
message_type	CRITICAL
monitoring_tool	API
routing_key	home_security
state_message	3
state_start_time	Aug 07, 2017 11:36:42.000 MDT
timestamp	Aug 07, 2017 11:36:27.791 MDT

In this case the Door to the Garage opened at 11:36:27 MDT. This created a critical incident in VictorOps that sent me a push notification to my phone instantly. The door was then closed about 6 seconds later. At that point the paging policy I have setup was canceled and the incident auto resolved.

There are lot of other bits and pieces I built into the system. My next plan is implement moisture sensors in key locations in the house and send those along as well. I will probably use the VictorOps Transmogrifier to create a different escalation schedule for those!

All in all, a pretty fun project. I posted the code and schematics to GitHub if anyone wants to build it themselves!