

Justin Wickett
Duke Smart Home
Sunday, May 25, 2008

CONTROLLING ELECTRONICS VIA SMS

By leveraging freely available technologies, an individual can use cell phone text messaging to remotely control electrical devices. As a result, the mobile phone's ability to serve as a communication platform expands into the electrical domain. With soaring energy costs and a popular "green" cultural movement, demand has grown for energy management utilities that are capable of providing consumption metrics from remote locations.

Controlling electrical devices via text messaging (SMS) relies on four separate principles:

1. A computer can communicate with an electrical switch.
2. An electrical switch can communicate with a computer.
3. A cell phone can communicate with a computer via SMS.
4. A computer can communicate with a cell phone via SMS.

The ability for a computer to communicate with an electrical switch is critical to remote home automation and requires special hardware. In my demonstration, I used SmartHome INSTEON-compatible electrical devices. Each INSTEON device has a unique address, which allows for devices to recognize one another and communicate over an electrical or RF network. These addresses play a similar role to MAC (Media Access Control) addresses found on Ethernet devices.

In my demonstration, I plugged my computer via USB into an INSTEON PowerLinc Controller to communicate with an INSTEON SwitchLinc Dimmer switch. The PowerLinc Controller can be plugged into any standard 120 volt single phase electrical outlet on your local electrical network, whereas the SwitchLinc device must use a switch box with neutral wiring. Below are images of the two devices I used:



INSTEON SwitchLinc Dimmer
<http://www.smarthome.com/2476d-light-dimmer.html>



INSTEON PowerLinc Controller
<http://www.smarthome.com/2414u.html>

Computer operating systems communicate with the PowerLinc Controller via the HID (Human Interface Driver) driver. My setup was based on a Linux machine running Fedora Core 7. I compiled from source Bob Paauwe's iLink INSTEON scene management software, which included the iplc kernel module. Even though I installed the iplc kernel module on my machine, I was told that it was no longer needed due to native HID support in the Linux kernel. I then verified that my operating system was aware of the INSTEON PowerLinc Controller by viewing `/var/log/messages` (run `"tail -f /var/log/messages"` via command line while plugging the USB cable from the PowerLinc Controller into your computer to see if the device was properly recognized). After having done that, I installed the development libraries needed to compile Bob's iLink INSTEON scene management software (run a `"yum install libusb-devel libglade2-devel gtk2-devel"`) and carefully followed the README and INSTALL instruction files he provided to install Link (the CLI version of his software) on my Linux machine. I was able to set up a simple scene configuration file by using Bob's template (see `sample_scene.inf`) and filling in the `"xx.xx.xx"` with the unique address from each of my INSTEON devices. After configuring my scene file, I ran (as root) `"/link -x all"` to propagate the changes and program devices to make the scene active. Once the scene was active, I compiled the `icmd` program in the utilities directory. The `icmd` program allows you to send direct commands without the need for links. By running `"/icmd xx.xx.xx ON 255"`, I was able to turn on the INSTEON device with address `xx.xx.xx`. When the switch has been successfully turned on, a response message is returned. As a result, the electrical switch communicates its status back to my computer.

The next step involved getting my cell phone to communicate with my computer. Most of today's mobile phones are Bluetooth enabled and capable of sending and receiving text messages. I never considered using my cell phone to call into a PBX (private branch exchange) server such as Asterisk running on my computer due to the complex nature of the software involved. Despite standard Bluetooth support in the Linux Kernel, I chose not to use Bluetooth as a means of communicating with my computer from my phone due to limited range issues. Had I used Bluetooth, I would have been unable to remotely controlling electrical devices from long distances. Relying on text messaging capabilities seemed to be my best option for long distance remote control over my electrical network.

In order to communicate with my computer via text messaging, I had to leverage functionality provided by Twitter.com's free service. Twitter is a SMS portal/gateway amongst many other things. Twitter stores all text messages sent to its short code phone number (40404) in a database, and displays them on the Web at a specific URL. By polling that specific Web URL from my computer, I could detect and analyze any changes. As a result, my cell phone was able to communicate with my computer.

Furthermore, communication between a cell phone and a computer can be bi-directional. My computer could communicate back to the cell phone via several methods. Most of the major cell phone service providers offer free SMS public gateways. As a result, an email can be sent to a public SMS gateway, which then forwards the contents of the email to a user's cell phone in text message format. A list of available public SMS gateways sponsored by major service providers can be found at http://en.wikipedia.org/wiki/SMS_gateways. Alternatively, Twitter can also be used to send cell phone notification updates should a user choose to accept them.

I only had to write a few lines of code to pull all of these technologies together so that they would work with each other. The following code is a very rough implementation that I originally used to test the feasibility of this idea. I plan on incorporating Bluetooth support as well as confirmation notifications once I get back to Duke University. Right now, I am using my cell phone to send Twitter public updates that are broadcasted out to all of my friends. This method is not secure, and spams your followers with updates about your electrical network's condition. I recommend creating a private account for testing purposes, or better yet using Twitter's direct messaging functionality.

Finally, this code relies on the Summize.com search engine, which parses and indexes every public message sent to Twitter. I could not poll Twitter.com because of rate limiting issues. My code polls Summize's REST API (which is simply a Web URL) every second checking to see if there has been an update. This polling method is not efficient and taxes Summize's servers. I recommend subscribing to and parsing Twitter's Pub Sub Jabber feed (see <http://groups.google.com/group/twitter-development-talk/web/jabber-pubsub>).

Below is my quick and dirty Python code that can be easily ported over to other languages:

```
#!/usr/bin/python

#Copyright 2008 Justin Wickett
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#You should have received a copy of the GNU General Public License
#along with this program. If not, see <http://www.gnu.org/licenses/>.

import feedparser, os, time #The 'feedparser' library can be installed from http://www.feedparser.org/

#TODO: Replace the username as well as the INSTEON address with the appropriate values
#TODO: Also make sure that the path is correctly set in the system command to icmd's path on your local machine

lastTweet = 0 #Used to keep track of the last Tweet received to make Summize queries less expensive
username = "xxxxxxx" #Twitter username who is sending the commands to control the electrical network
insteonAddress = "xx.xx.xx" #INSTEON address of the device you want to turn on and off

while(1):
    feedUrl = "http://summize.com/search.atom?q=from%3A"+username+"&since_id="+str(lastTweet) #Polling Summize
    feed = feedparser.parse(feedUrl)
    if len(feed['entries']) > 0 and feed['entries'][0].link.split('/')[-1] > lastTweet:
        if cmp(feed['entries'][0]['title'], "Bedroom lights on") == 0: #Check for the "ON" command
            os.system("icmd "+insteonAddress+" ON 255") #Turn the lights controlled by my switch on
        if cmp(feed['entries'][0]['title'], "Bedroom lights off") == 0: #Check for the "OFF" command
            os.system("icmd "+insteonAddress+" OFF 255") #Turn the lights controlled by my switch off
        lastTweet = feed['entries'][0].link.split('/')[-1] #Save the last Tweet so we aren't stepping over ourselves
        time.sleep(1) #Sleep one second, and execute code again
```