Portable RFID Tag Scanner

Ethan Pfenninger

Sponsor - Phil Collins

UNHCEMS®

UNH CEMS

UNH CEMS is an all-in-one platform designed by the Office of Environmental Health and Safety and the Research Computing Center. It is a chemical inventory system for institutions that provides the ability to easily track, manage, and order chemicals for labs. Additionally it allows first responders the ability to determine what chemicals are in a particular building and what safety precautions need to be taken in case of an emergency.

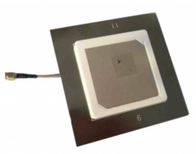
The Problem

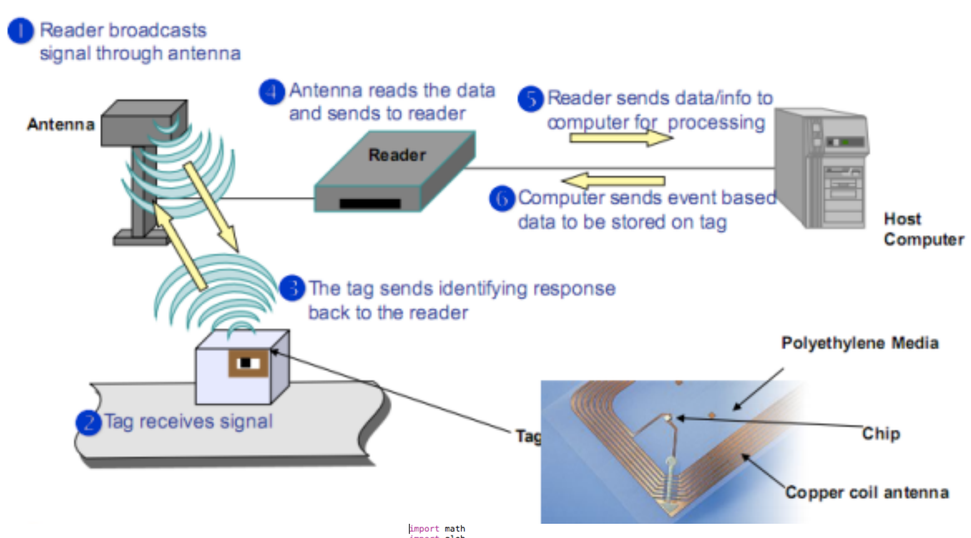
UNH CEMS routinely has to go through their inventory of chemicals to ascertain the quantity and location of all substances. This process is currently done using a barcode system, with all packages being scanned by hand and inputted into a database. With an inventory of over 30,000 chemicals, this process is both time consuming and imprecise. By the end of the inventory operation the information is out of date. This leads to issues in the case of a fire or other catastrophic event where the response would be determined by what substances were involved.

Solution

I used a Raspberry Pi minicomputer, hand portable antenna, and a rechargeable battery pack capable of powering the device for an extended duration. This device is capable of going into a room, scanning all tags at a distance of up to a meter and a half, and adding the information to a database. Line of sight is unnecessary, significantly decreasing scan time and allowing for more accurate and up to date inventories. The open source nature of the project and easily obtainable hardware allow easy incorporation by other institutions while maintaining a price point significantly below other comparable solutions.

1	19	4530-3031-0000-0000-0000-0000
1	10	3008-33B2-DDD9-0140-0000-0000
1	11	0013-2400-0000-0000-0000-4D9E
1	3	A5A5-1001-0500-0000-0000-B6C9
1	7	0000-0000-0000-0000-0A10-0489
1	4	6666-8888-3333-1111-0000-BEC9
1	4	0000-0000-0000-0000-0A10-3565
1	4	0000-0000-0000-0000-0A10-3564
1	3	0000-0000-0000-0000-0A10-0490
1	4	0000-0000-0000-0000-0040-1306





RFID Tag

Radio Identification

RFID Antenna

UART interface

RFID Translator Chip

USB connection

Rapberry Pi

Wireless / USB

UNH CEMS DBMS

```
import generator
import RFID

def main():
    print generator.returnEpcCode()

    ann = RFID.AntennaDevice()

    print "Firmware info: %s\nHardware info: %s" % ann.get_system_info()

    print "Activating antenna"
    ann.set_antenna_state(True)

    print "Tags:"
    for epc, rssi in ann.iter_epc_rssi():
        print epc.encode("HEX"), rssi

    ann.set_antenna_state(False)

if __name__ == "__main__":
    main()
```

```
class AntennaDevice(object):
    def __init__(self):
        self._comm = AntennaDeviceCommunicator()

    def get_system_info(self):
        get_info_payload = lambda payload: self._comm.send_and_receive_output(0x10, payload)
        firmware_info = get_info_payload("\x00")
        hardware_info = get_info_payload("\x01")
        return firmware_info, hardware_info

    def _change_freq(self, is_hopping, freq_khz, rssi_threshold=-40):
        if is_hopping:
            # Add frequency to list of hopping frequencies
            mask = 0x40
        else:
            # Clear the list and add only this frequency (not hopping)
            mask = 0x80
        payload = struct.pack("B", mask)
        # Only 3 bytes of frequency
        payload += struct.pack("<I", freq_khz)[:NUMBER_OF_FREQ_BYTES]
        payload += struct.pack("b", rssi_threshold)
        self._comm.send_and_receive_output(0x41, payload)

    def set_profile(self, start_freq_khz, end_freq_khz, freq_increment_khz, rssi_threshold):
        is_hopping = False
        for freq in xrange(start_freq_khz, end_freq_khz, freq_increment_khz):
            print "Adding %d to frequency list" % (freq,)
            self._change_freq(is_hopping, freq, rssi_threshold=rssi_threshold)
            is_hopping = True
```

```
import math
import glob
import struct
import serial

HEADER_LENGTH = 2
EPC_OFFSET_IN_COMMAND_RESULT = 8
NUMBER_OF_FREQ_BYTES = 3
TTY_USB_WILDCARD = "/dev/ttyUSB*"
UART_BAUD_RATE = 115200


def parse_rssi_i_q(rssi):
    rssi_i = (rssi & 0x0f) * 2
    rssi_q = (rssi >> 4) * 2
    return rssi_i, rssi_q, math.sqrt((rssi_i ** 2) + (rssi_q ** 2))


class AntennaDeviceCommunicator(object):
    def __init__(self):
        tty_over_usb_devices = glob.glob(TTY_USB_WILDCARD)
        assert len(tty_over_usb_devices) == 1, "Card not found or multiple cards connected"
        self._serial_conn = serial.Serial(tty_over_usb_devices[0], UART_BAUD_RATE)

    def send_command(self, command, data):
        command_header = struct.pack("BB", command, HEADER_LENGTH + len(data))
        command_buffer = command_header + data
        self._serial_conn.write(command_buffer)

    def receive_output(self):
        header = self._serial_conn.read(HEADER_LENGTH)
        command, length = struct.unpack("BB", header)
        output_buffer = self._serial_conn.read(length - HEADER_LENGTH)
        return command, length, output_buffer

    def send_and_receive_output(self, command, data):
        self.send_command(command, data)
        output_command, output_length, output_result = self.receive_output()
        assert command == output_command - 1, "Command result is for an unknown command: %d" % (output_command - 1,)
        return output_result
```