Smart Bracelet 101

Developer's Guide

Embedded Team

Michael Lishansky
Valentin Dashinsky
Sapir Cohen

Table of Contents

Introduction

The objective of this project was to create a prototype for a “smart bracelet” to be used by the IDF medical corps. The Bracelet is fitted to wounded soldiers, and used to scan treatments using NFC. Each treatment has an NFC tag attached, and the tag is scanned and stored by the bracelet.

The bracelet interfaces with an Android app installed on an Android phone or tablet. The interfacing is done using the specific Bluetooth protocol defined in this document.

The primary objective was ease of use. There are no buttons or other control interfaces on the bracelet. All that is required to record a treatment, is to “touch” the bracelet with the treatment item. This allows the medic to focus on treating the wounded, instead of wasting valuable time on recording treatments manually.

The programming language used in this project is C/C++.

Note: the Arduino has no persistent writable memory. This means that all changes, including the tag database, are lost as soon as it is turned off. According to the current design, the tag database would be transferred to an Android device and saved there. If you wish to save the database persistently on the Arduino, a possible solution would be to add an SD card component to the Arduino, and save the database on the SD card.

Setup Instructions

Hardware

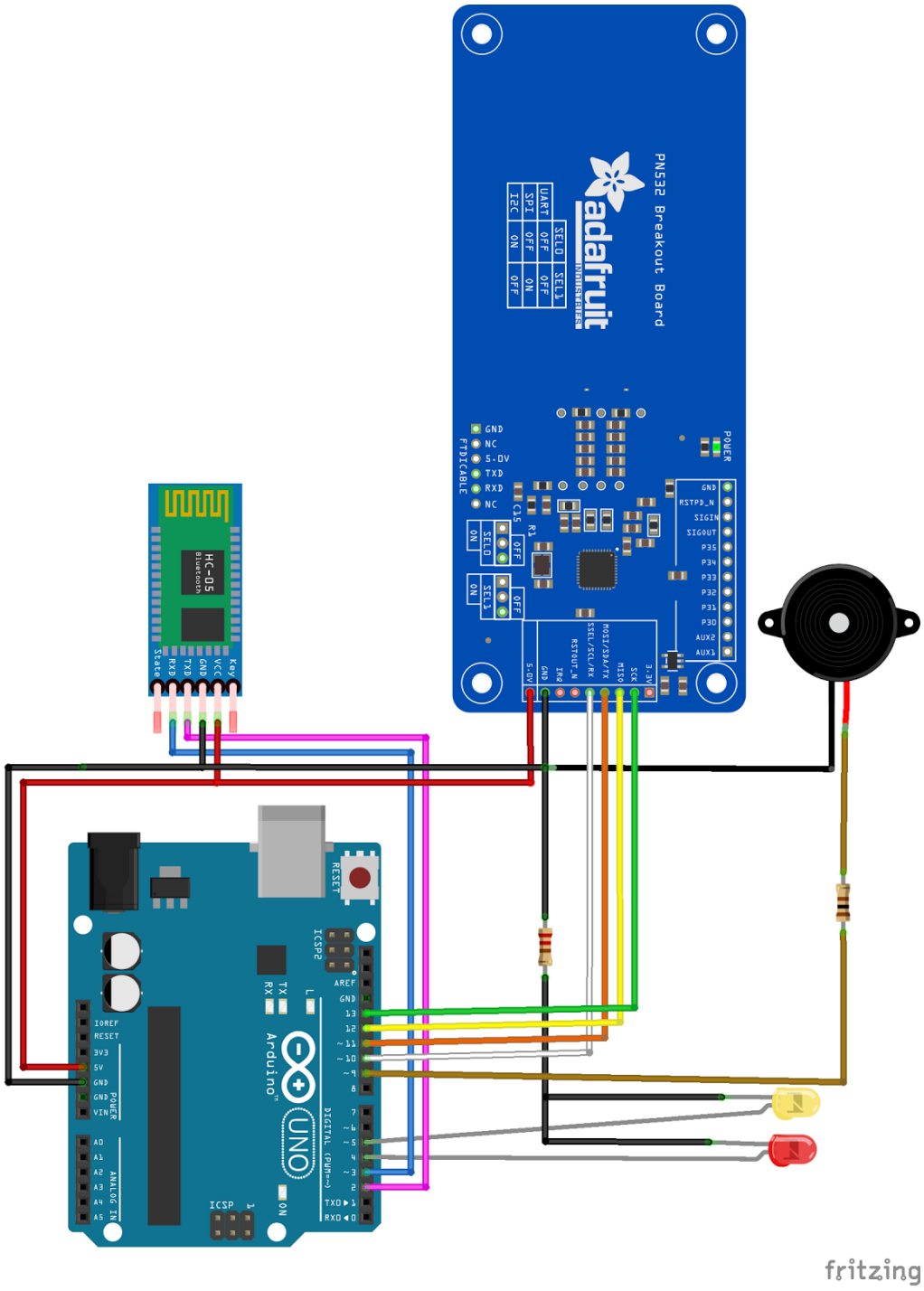
The hardware is comprised of the following:

- Arduino (MCU)
- PN532 (NFC scanner)
- HC-05 (Bluetooth chip)
- Yellow/Green LED (indicates success)
- Red LED (indicates an error)
- Buzzer (simple speaker that is used to play error/success tones)

Testing was done on Arduino UNO, Arduino Pro Mini and Arduino Nano.
The hardware should be connected as described in the following schematic.

Note: It's possible to add an NFC tag with the HC-05's MAC address written inside the tag's memory (as a string). It is then possible to read that tag using the Android device's NFC scanner. This will change the screen to the relevant bracelet in the Android app.
This allows the Android user to easily select the bracelet in front of him (instead of sorting through the list of bracelets and trying to find the relevant one), simply by touching it with the Android device.
Important: this attached NFC tag must be located far from the PN532 NFC scanner. Otherwise the scanner will constantly read this tag. Therefore, it is recommended to attach it to the opposite side of the scanner (the "bottom" of the bracelet).

Schematic Diagram



Note: the pin numbers are the same for all tested Arduino types.

The Fritzing file of this diagram can be found on the GitHub repo.

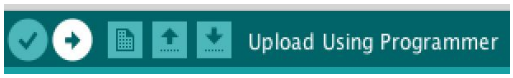
Drivers

The Arduino Uno and Arduino Nano have a USB port, and drivers for them are installed with the Arduino IDE.
The Arduino Pro Mini doesnt have a USB port and needs a USB-Serial converter (for example FTDI), more on that can be found here:
http://lab.dejaworks.com/programming-arduino-mini-pro-with-ftdi-usb-to-ttl-serial-converter/
Drivers for FTDI USB-Serial converter can be found here:
http://www.ftdichip.com/Drivers/D2XX.htm
Since the project is hosted on Github in a Git repo, it's recommended to use Git:
https://git-scm.com/
It's also possible to just download the project files as a zip from the repository:
https://github.com/ValkA/BraceletIOT (click on "Clone or Download" -> "Download ZIP").

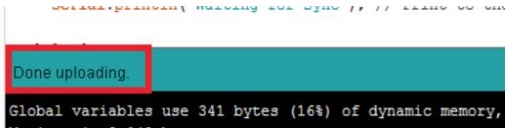
Setting up the workspace

The project was built using Arduino IDE 1.8.2.

1. Download the IDE from here https://www.arduino.cc/en/Main/Software and install it
2. Clone the project "git clone https://github.com/ValkA/BraceletIOT.git"
3. Copy the NFC libraries from "Libraries_for_IDE" (GitHub repo) into "Documents\Arduino\libraries" (your PC).
4. Open "Bracelet_Arduino/Bracelet_Arduino.ino" with Arduino IDE.
5. Connect the arduino board to USB.
6. From "Tools > Board" choose your board (Mini or Uno).
7. From "Tools > Port" choose the COM Port of your Arduino.
8. Click on the upload button.



9. After compiling and uploading, you should see "Done Uploading" at the bottom.



Project Structure

The repo can be found here https://github.com/ValkA/BraceletIOT
In the folder "BraceletIOT/Bracelet_Arduino/" we have:

- Bracelet_Arduino.ino - The "main" file. Contains the setup and main loop functions. It uses the NFC Library and handles the logic of when to listen to the NFC scanner, and when to respond to Bluetooth or Debug commands.
- Logs_Container.h - The main data structure in the project - defines how all information is stored in the Arduino's limited memory.
- Notes.h - contains defines of various notes frequencies and the Tone structure. Contains the logic for sounds and blinking LEDs.
- Parser.h - contains parsing functions that handle input/output. Parses tag scans as well as the Bluetooth protocol.
- TagReader.h - contains a class that handles communication with the PN532 NFC scanner.

The folder "/BraceletIOT/Libraries_for_IDE" contains NFC libraries that should be copied to "Documents\Arduino\libraries" in the developer's PC.

Memory constraints

The Arduino RAM is comprised of only 2kB, and the objective of this project was to use it for storage. This required some special considerations for memory management.

It's important to note that the Arduino's 2kB are also used for function calls on the stack, as well as temporary variables. This means only about 1kB are available for storage. Therefore, it is imperative to reduce the size of each record[1] stored in the Arduino as much as possible. We settled on the following format:

- 14 bits for the timestamp.
- 4 bits for the record type (allows for 16 types, which are detailed in the next section).
- 28 bits for the data.

For a total of 46 bits per record.
Note that those are bits and **not Bytes**.

To achieve those goals, we used C Unions and C bitfields liberally. It's important to acquaint yourself with those concepts before continuing work on this project.

More information can be found in the internal documentation of the Logs_Container.h file.

Note: since this is an embedded project with a very restricted memory pool, it is highly inadvisable to use dynamic memory allocation. Therefore, all memory allocation in this project is static, and is done at boot time.

Currently the bracelet is configured to store a maximum of 100 records. It is possible to edit this defined number in the Logs_Container.h, but higher numbers were not tested and are likely to result in instability.

[1] The next section details the possible types of records. Each command detailed there is stored in the memory (unless specified otherwise) and constitutes a single record (unless specified otherwise).

Bluetooth protocol

Generic Commands

The bracelet reads commands from the Bluetooth serial in the format <$type,$data>.
After receiving a command, the bracelet sends '#' as an ack. If there wasn't enough memory to add a record, the bracelet will send '!' back, instead of '#'.

The following record types can be sent to the bracelet:

- <0,Integer> adds a tag scan. In case the doctor wants to add a treatment manually through the Android device. $data is the treatment number.
- <2,Integer> - adds new data (such as temperature measurement by doctor).
- <3,$time,$tsid,Integer> - update record [$time,$tsid] with new data (Integer 14bit max)
- <4,Integer> - adds a record of headquarters communication (such as evacuation notification), $data can be any kind of Integer.
- <5,Integer> - adds a record of blood pressure. $data = blood pressure
- <6,Integer> - Turns on buzzer, $data is not used, but should be some kind of Integer. 0 can be put there.
- <7,$time,$tsid,0> - delete record represented by $time,$tsid.
- <8,Integer> - adds soldierID record. $data should contain the soldier's personal number.
- <13,Integer> - adds soldier status (severity of injury). $data is id of the status.

For example, the command <1,123> will add a record into the bracelet that represents a new connection by an Android device that is represented by the number 123. As a response you will get the records database.

Special commands

- LOCATION: To register location you should send <10,(double/float)latitude><11,(double/float)longitude> you will see the location when you will send <1,doctor_id>. the location will be among the tags in the container. for example, [<1,3,0,1><10,32.2343><11,7.23456>]. fields 2 and 3 represent the latitude and longitude. Note that Location is comprised of 2 records.

- SYNC: <14,Integer> - a special record that can be used to keep the Android and the bracelet in sync. When the bracelet receives this record, it sends the entire database back, but does not place this record in it's database. The format of the database is the same as in <1, Integer>. This allows this record to be sent periodically from Android to the bracelet to make sure they are both in sync (for example, every 30 seconds). Similar to <6, Integer> the $data here is not used, but should be some kind of Integer.

- DOCTOR CONNECTION: <1,Integer> on connection this is the first thing should be sent to the bracelet. $data represents the unique number of the doctor (could be his personal number). As an acknowledgement the bracelet will send back its database with the following format: [<$type,$time,$tsid,$data>, <$type,$time,$tsid,$data>, ..., <$type,$time,$tsid,$data>] also the format <3,$time,$tsid,$pointer_time,$pointer_tsid,$data> may be in the container, which represents an update record (that was added with <3,$time,$tsid,Integer> message from doctor).

Note: as explained previously, there can be 16 possible message types. Type number 12 is the only one not currently used (type 15 is used internally). If you wish to increase the number of possible message types, it is possible to increase the amount of bits allocated for the “type” field in each record from 4 to 5 (or any other number). This can be done in the Logs_Container.h file.

Tag Format

The tags scanned by the bracelet must be in one of the following 2 formats:

- Integer String
  For example: 50 Optalgin 10mg
  This format is used for different treatments. It's supposed to be in an NFC tag attached to the treatment item, to be scanned by the bracelet.

- dInteger String
  For example: d1234567 Israel Israeli
  This format is for the soldier personal number. It's supposed to be in an NFC tag attached to the soldier's tag (diskit).

Notice that in both cases **only the number is scanned by the bracelet**. The following string is completely ignored. It is there optionally so that it could be used in other devices.

The Integer must be smaller than 2^28, which is the maximum number that can be saved in the bracelet (28 bits are stored for each record's data).

Debugging

It is possible to send debugging commands to the Arduino through the Arduino Serial (can be accessed via “Tools > Serial Monitor” in the IDE.
There are currently 3 types of debugging commands:

- d - print the current database to the Serial.
- m - print memory information to the Serial (number of records currently stored)
- <$type,$data> - add a specific record to the Arduino. The record would be parsed in the exact same manner as a record sent to the Arduino from Bluetooth.

If the record is not recognized as one of the above legal records, you'll get an error and the record would not be added.

The Arduino also sends debugging information to the Serial during normal operation, such as detailed error messages. Therefore, if you encounter unexpected behaviour when interfacing through Bluetooth, connect the Arduino through the serial to see detailed debug messages.

Sounds and LEDs

You can change the buzzer sound or LED lights using a Bluetooth terminal.
The notes are configured for different cases and you can change configuration of any type separately.

The types are:
TurnOnSuccess = 0
TurnOnFailed = 1
ScanningSuccess = 2
ScanningFailed = 3
ConnectingSuccess = 4
NewAppMessage = 5
BeepFromApp = 6
UnknownTag = 7

The command should starts with '[' and end with ']'.
To change a sound insert: [0, type, frequency, freqParam , delay, repeats]
*"type" is between 0-7 as explained above.

For example: [0,2,31,50,200,6] - will change the sound for ScanningSuccess.
To change LED i insert: (i={1,2}) [i, type, delayOn, delayOff, repeats]
*Delay is in ms.
For example: [0,2,500,500, 3] - will change the first/second led blinks for ScanningSuccess.