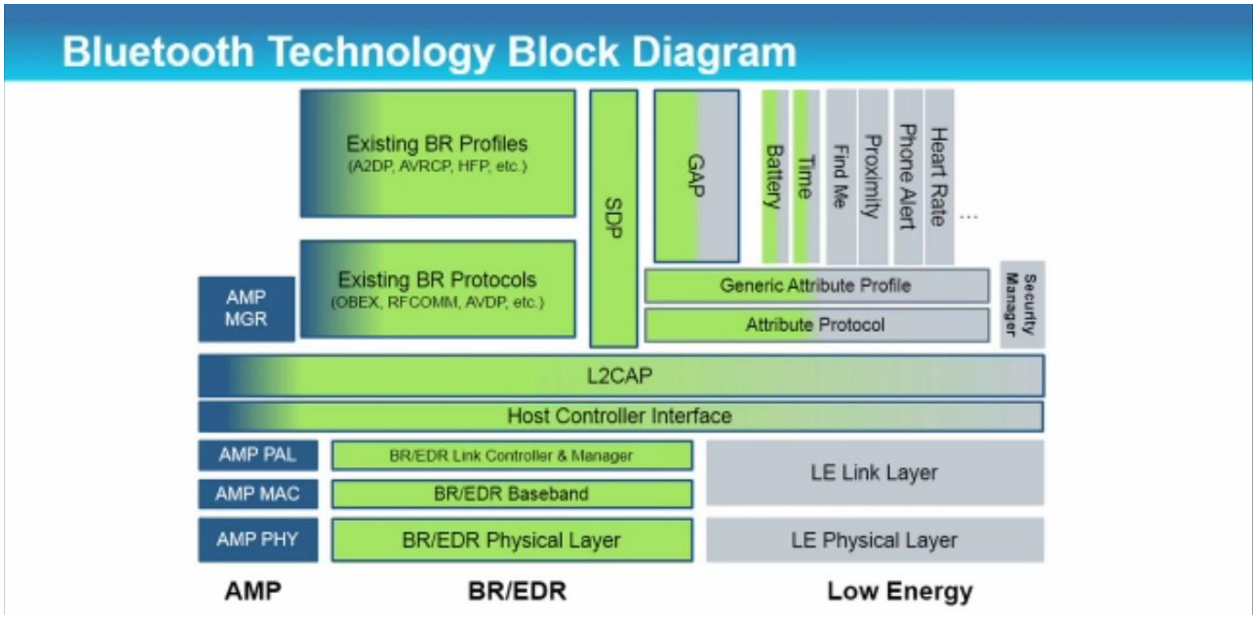
GATEBOX: SOFTWARE INSTRCUTIONS

By Shafeeq Rabbani

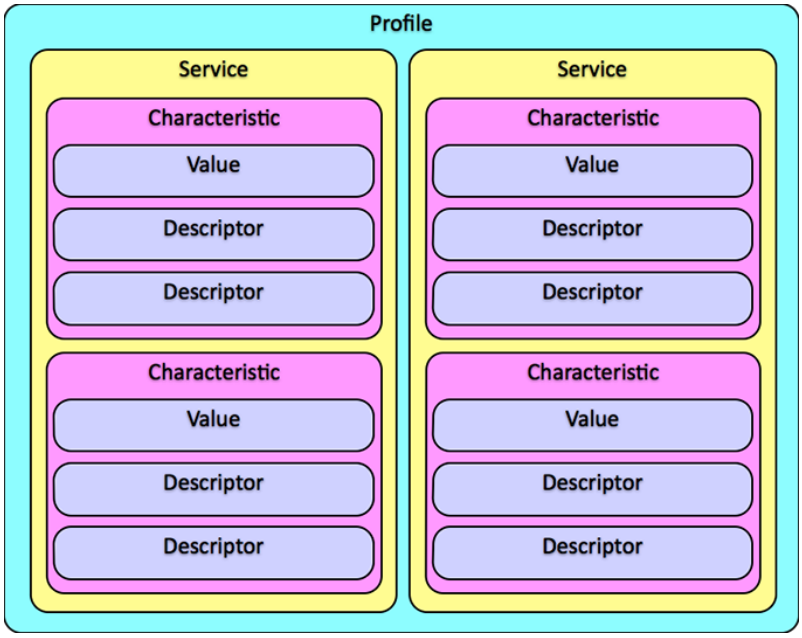
Contents

GATT PROFILE

It is essential that in order to fully understand this project, you study what is GATT profile. GATT profile, short for Generic Attribute Profile, sits on top of the attribute profile. It is used in Bluetooth Low Energy. Look at the image below to get an idea of the



Much like how BACnet BIBBs contain properties, GATT characteristics contain properties. Look at the diagram below for a broad understanding of the GATT structure.



nRFgo STUDIO

NOTE: if you don't want to use nRFgo STUDIO, you can simply use the services.h file already created for you found with the Arduino code.

In the GATT Profile, Services and Characteristics are described by what is called the UUID (Universally Unique Identifiers). We made custom UUIDs:

Table A:

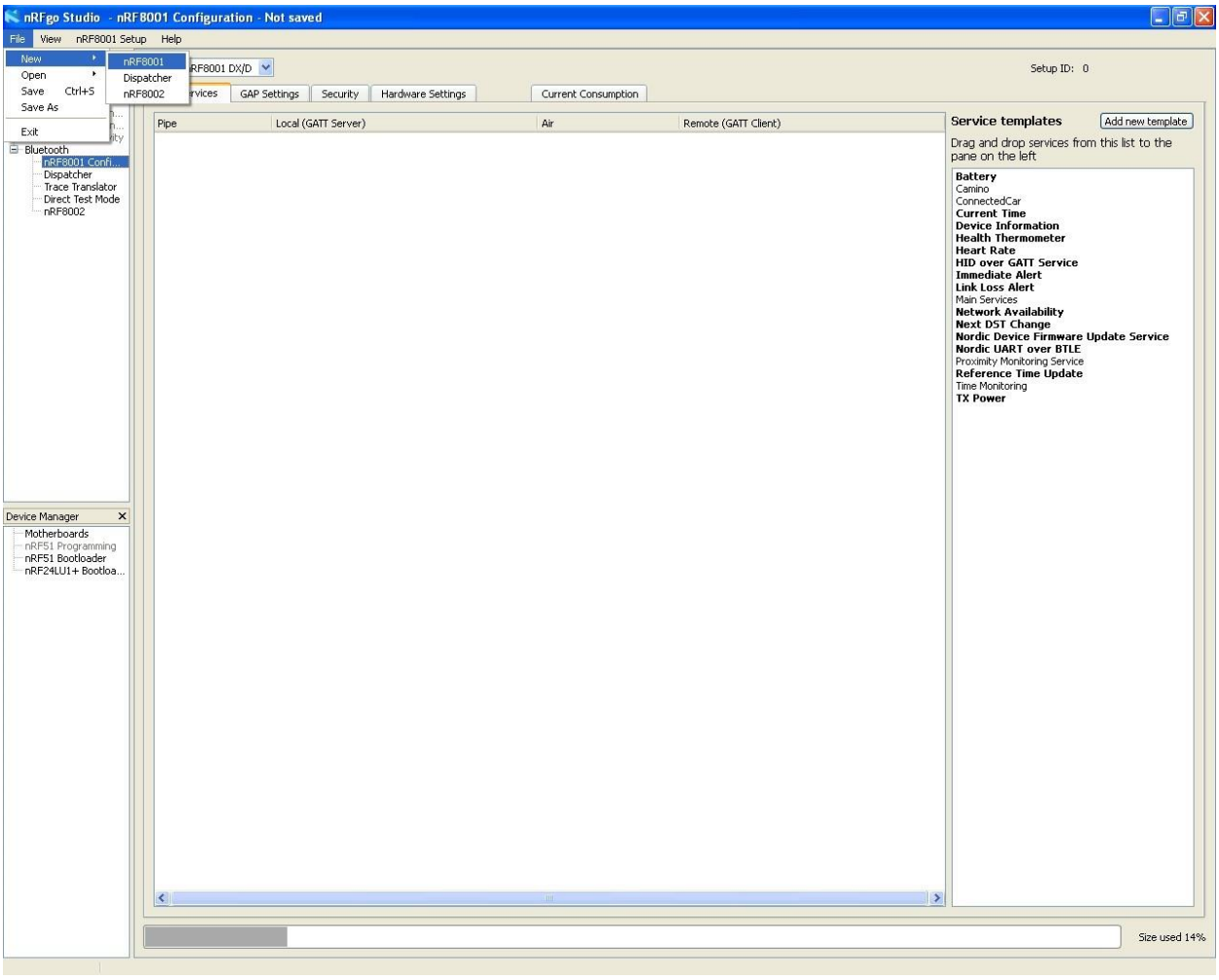
Service Name	**UUID**
ConnectedCar	CC01

Table B:

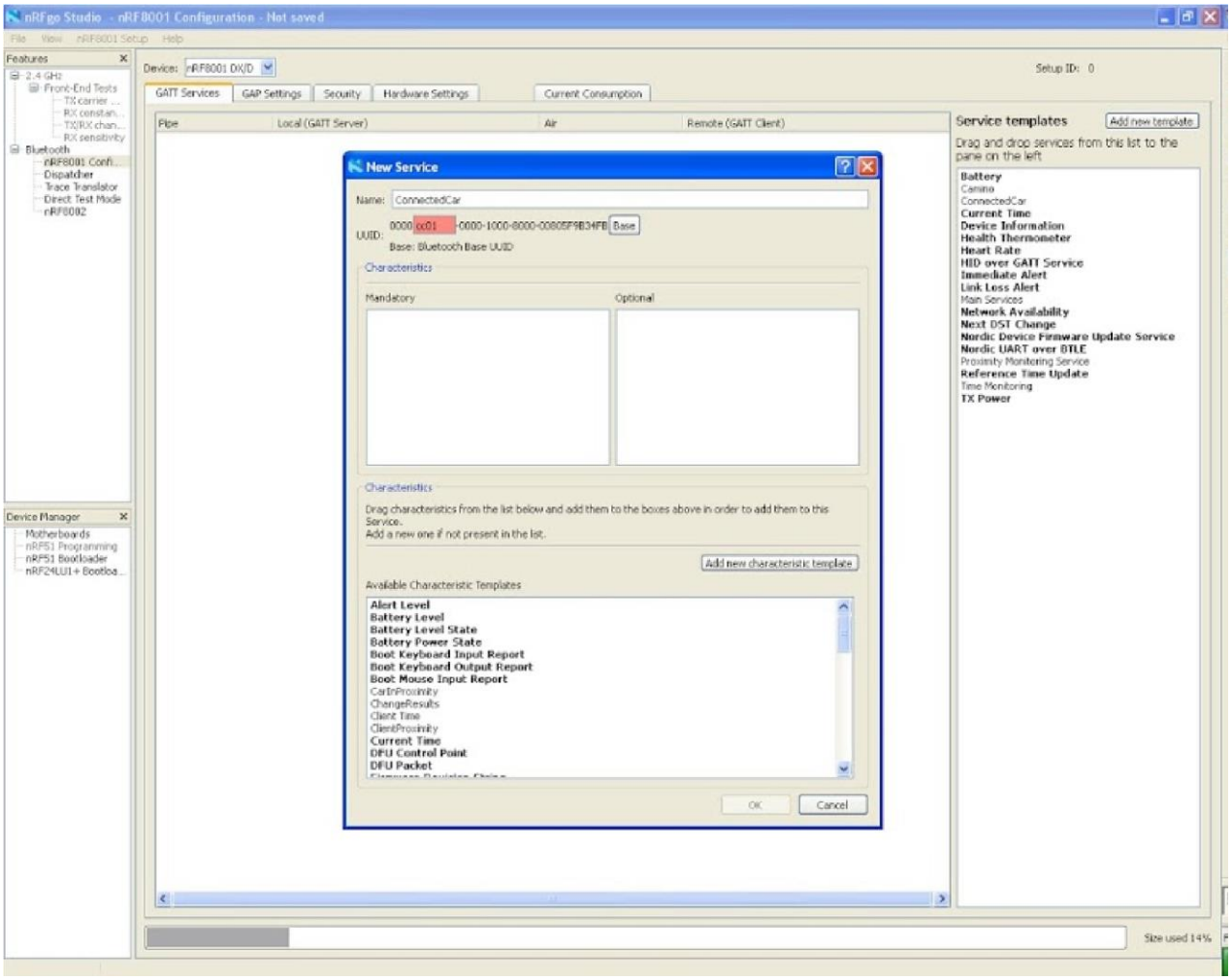
Characteristic Name	**UUID**	**Possible data accepted**	**Max data length**
Password	CC02	4-digit decimal number	2
changePassword	CC03	4-digit decimal number	2
Result	CC04	1-digit decimal number (1 or 0)	1
changeResult	CC05	1-digit decimal number (1 or 0)	1
lightsOn	CC06	1-digit decimal number (1 or 2)	1
garageDoor	CC07	1-digit decimal number (1,2 or 3)	1

You will have to build a GATT structure as described above. Different manufacturers have different software applications you can use to build your custom GATT structure. As we are using the nRF8001 chip, we will use the software that comes with it, nRFgo Studio. This software can be downloaded from the internet for free from: https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRFgo-Starter-Kit or you can extract it from the zip file in the "Software Needed" folder. While installing this software onto your computer, make sure to select the 32/64-bit version as per your operating system.

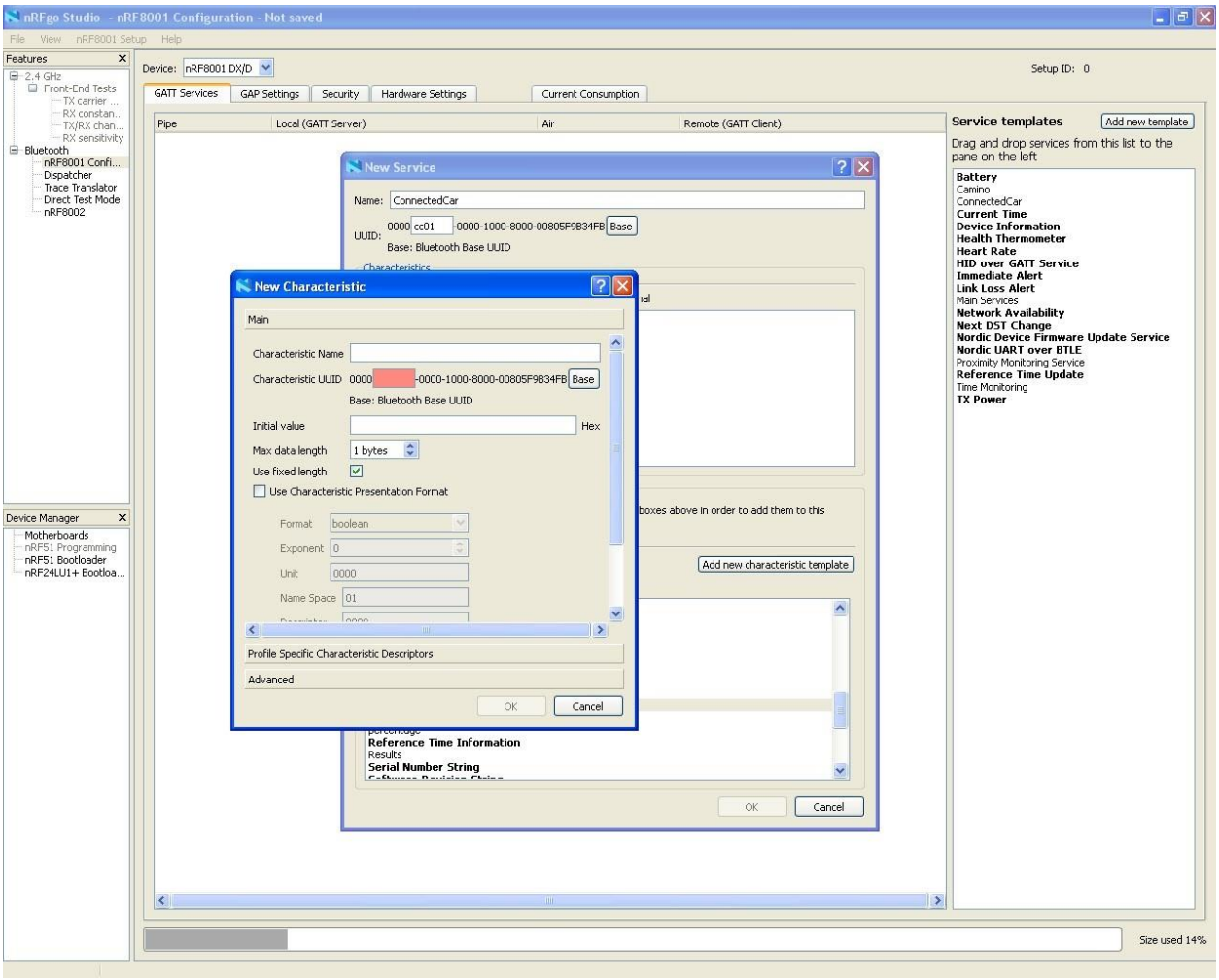
To make your GATT structure, click file → new → nRF8001.



Name the service and give it a UUID (Refer to Table A). UUID is in hexadecimal format.

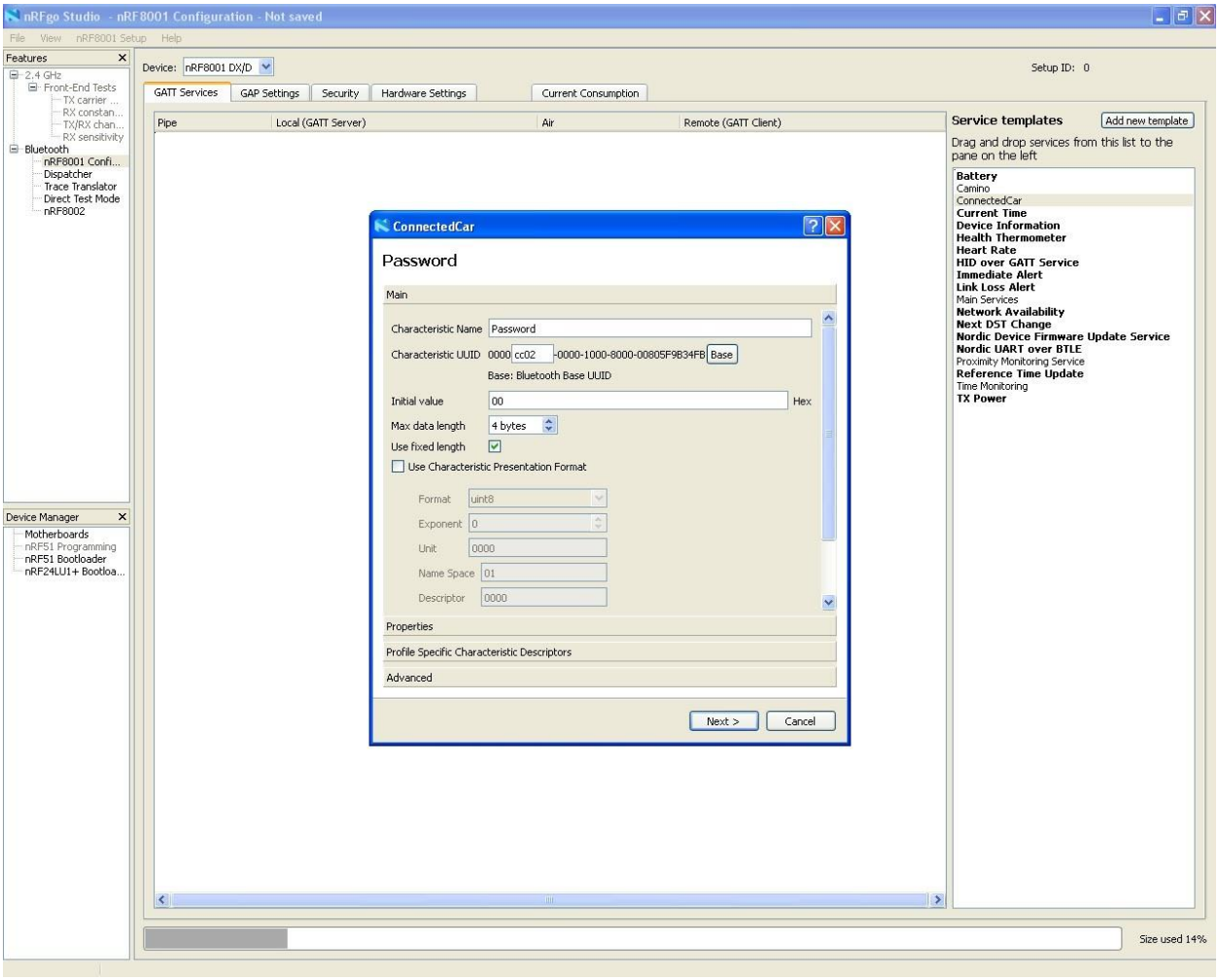


Click add new characteristic template. Name the characteristic, and the UUID associated with it and the byte. Initial value must be set to 0. Refer to Table B for the name, UUID and max data length of each characteristic.



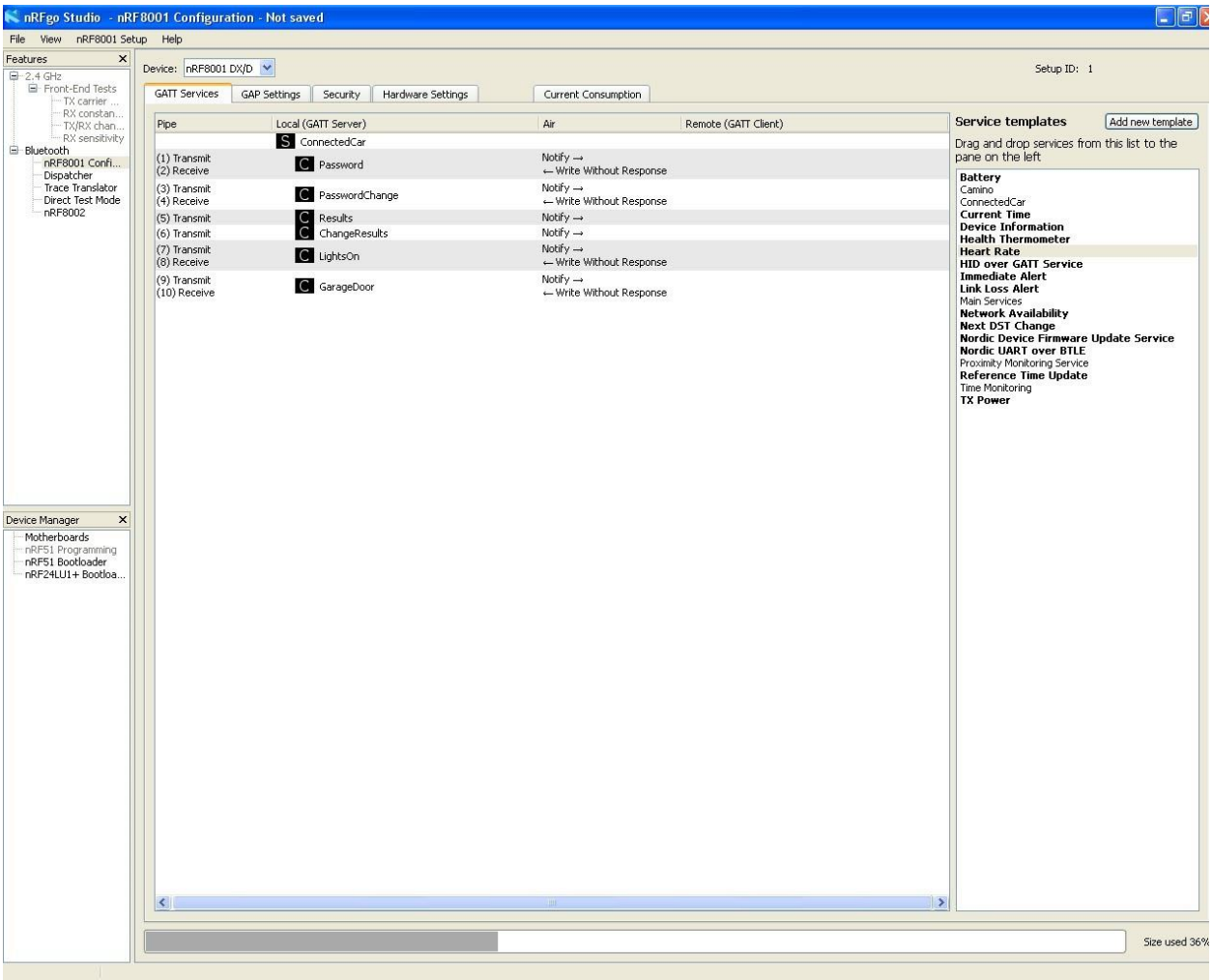
Click on profile specific characteristic descriptors and select Notify and Write without response.

(NOTE: Indicate is like Notify except with acknowledgement. Working with acknowledgement data is very complex and it is recommended that you avoid using it for now).



Click next and when the pop up window disappears, click and drag ConnectedCar under GATT server.

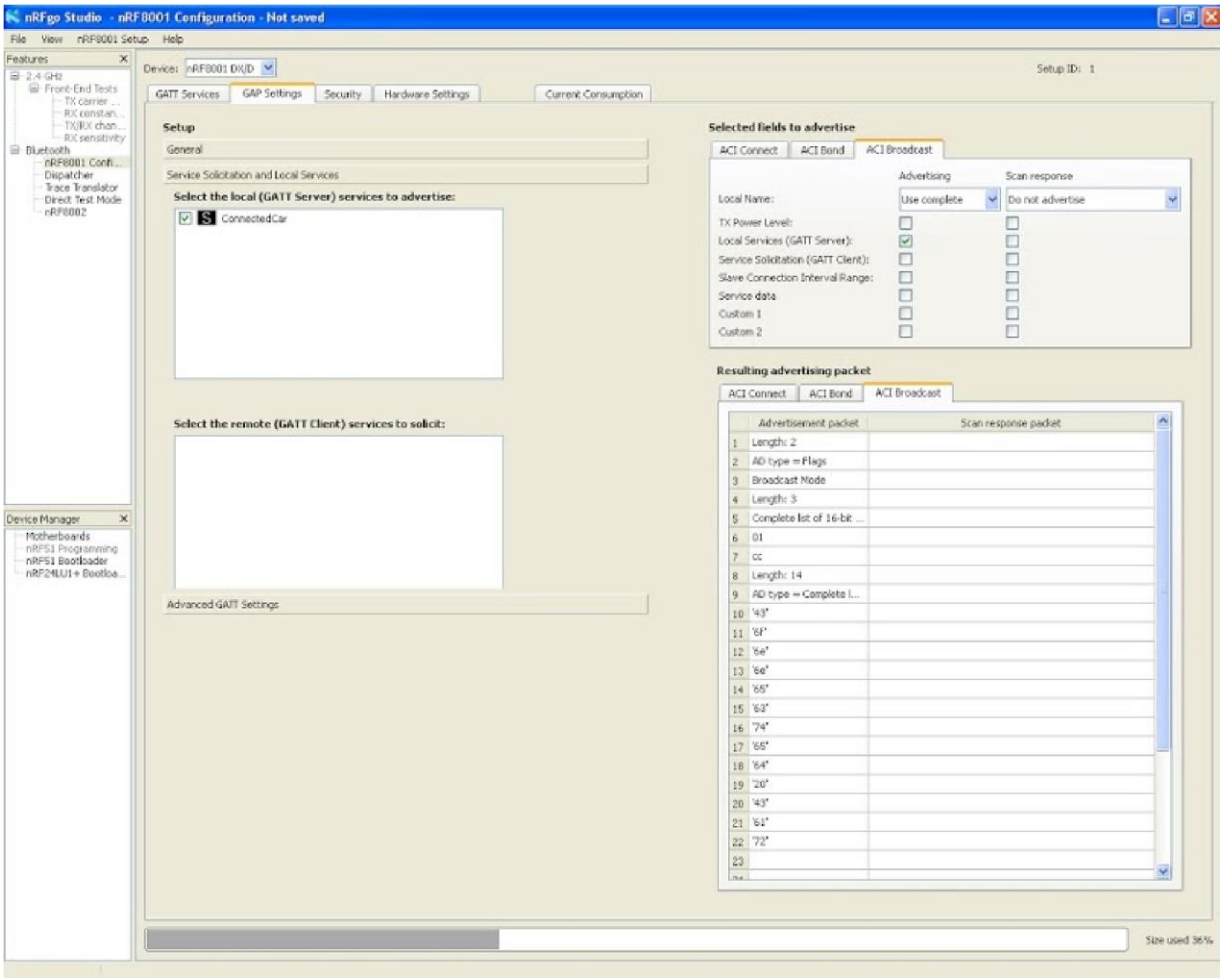
You should create all characteristics necessary so that your final window looks like this when you drag it. Refer to table A for characteristic codes.



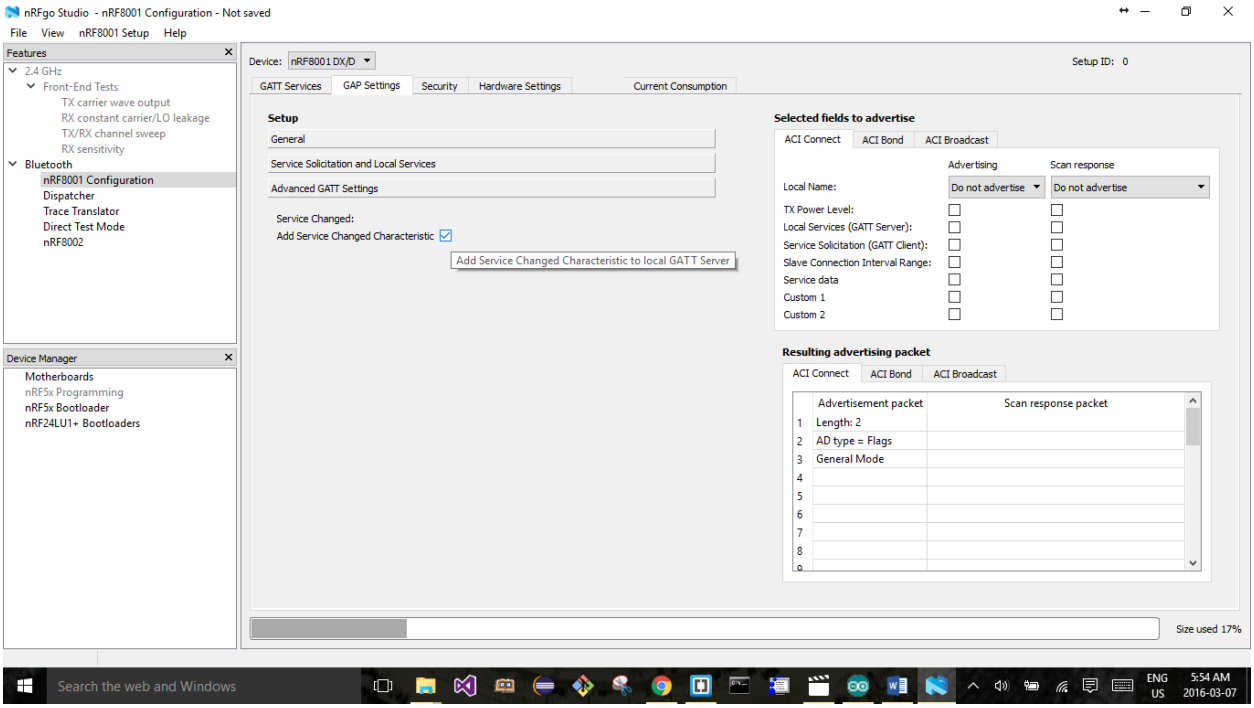
Click on GAP Settings.

Under service solicitation and local services, put the checkbox on the GATT server.

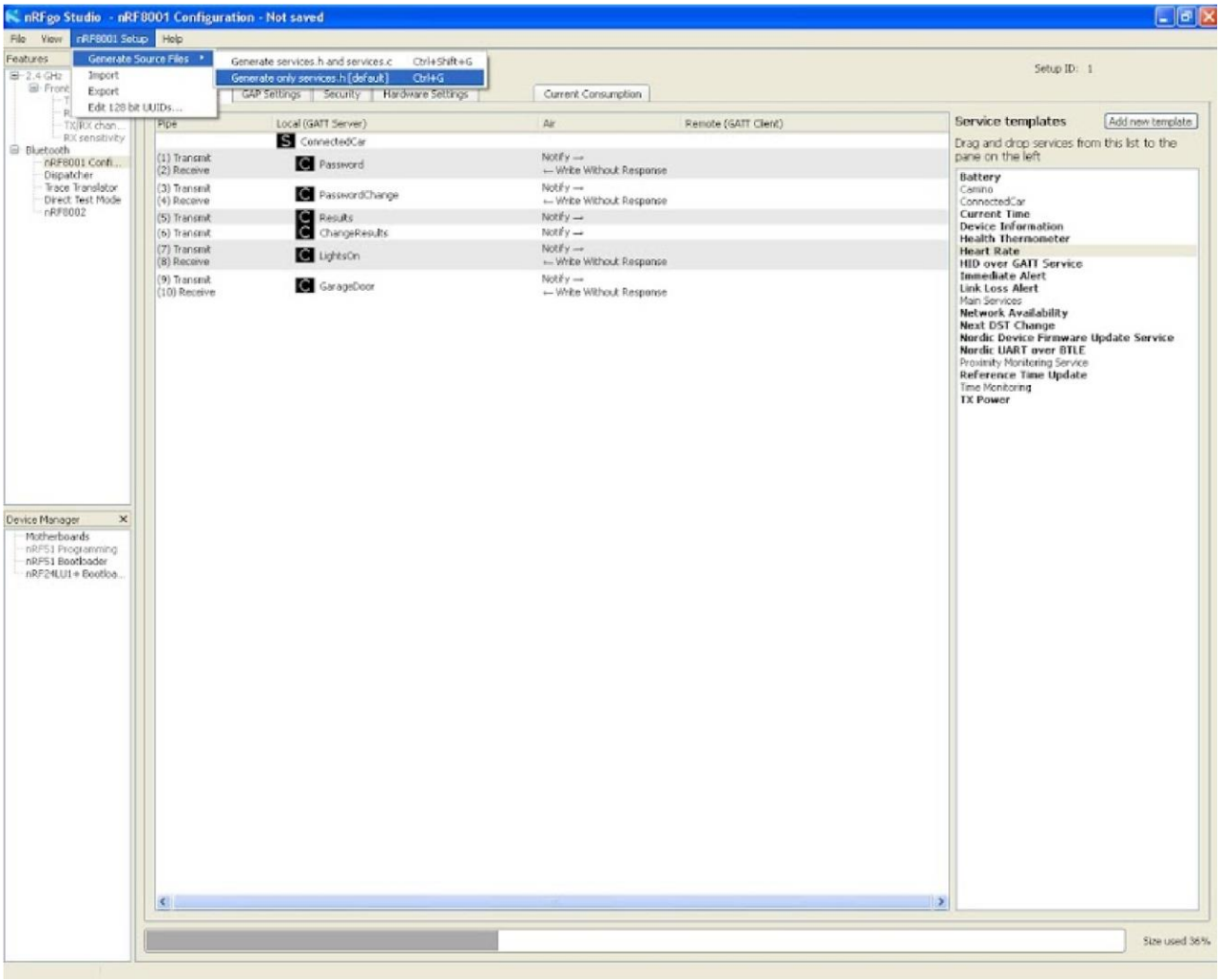
Click on ACI Connect, from the drop down menu select Use Complete and click the checkbox on local services (GATT server). Do the same for ACI Bond and ACI Broadcast.



Go to GAP Settings -> Add Service Changed Characteristic. This feature is useful if you are making your custom services and characteristics. Sometimes what tends to happen is that the older version of services and characteristics that you make are stored in the cache of your smartphone and are not overwritten even if you recompiled a different version of the services.h file onto the GATT server (the Arduino board is our GATT server). Service Changed Characteristic will allow the client to do service discovery every time it connects to the server.



Click on nRF8001 setup → generate source files → generate only services.h.



This will generate two files. services_lock.h and services.h. Do not use services_lock.h. This will permanently write to your UUIDs to nRF8001 chip. From then on, the services and characteristics nRF8001 chip cannot be changed. The Arduino code already has #include "services.h".

Download and Install Arduino IDE version 1.0.6

You must use version 1.0.6 because the code will not compile on later versions of the IDE. You can download this from the Arduino website or extract it from the GitHub folder "Software Needed".

Special note for detailed list: If your ATMEGA328p chip does not have the Arduino bootloader on it, then the bootloader must be uploaded onto the chip. There are various methods on the internet that show how to upload the Arduino bootloader onto an ATMEGA328p chip (For example, using the AVR ISP MKII). You can choose any method. You could also purchase an ATMEGA328p chip with the Arduino bootloader already onto it.

Install BLE and Time Library

If you have used Arduino IDE in the past, you would know how to add external libraries into the Arduino. Add the BLE library and Time Library.

Test the Arduino Code

In order to make sure your Bluetooth chip is working properly and that your wiring is correct compile and upload the ConnectedCarTest code into the Arduino UNO. The only difference between ConnectedCarTest and ConnectedCar is the fact that ConnectedCarTest has all the pins (except for digital pin 6) commented out. As initially you don't have all your hardware relays and pull up and pull down resistors connected, the voltage at a certain pin can fluctuate creating values that fill flood the serial monitor. This will make it difficult for you to test your code.

A good way to test if your UUIDs are working correctly is by an app on the apple app store called "LightBue Explorer – Bluetooth Low Energy" By Punch Through. This app is free to download. It will allow you to send specific values to a UUID. You can also test directly with the Windows Smartphone Application (which you would use).

Open the Serial monitor. If it asks you to remove pin 6 and reset, keep on resetting until the device is 'active'.

Open the app (while the serial monitor of the Arduino IDE is turned on) and enter UUID CC02. Enter 0000. It this password is correct (as in the EEPROM address bits 1000 to 1003) it will say "correct password". If it is incorrect, it will say "incorrect password, 2 tries remaining". It will also tell you what password is stored in the EEPROM. Enter this password into your favorite Decimal to Hexadecimal converter online to get the four-digit Hexadecimal code. This four-digit Hexadecimal code must then be written to CC02.

Note: if you decide to make your own customized GATT services and characteristics and find that the LightBlue app is showing an older version services and characteristics you made, simply turn off your Bluetooth and turn it on again. This will result in the phone getting rid of its Bluetooth cache and make a fresh start with service discovery of the GATT server.

Changing the password

Think of a four digit-number, insert it into your favorite Hexadecimal to Decimal converter online to get the hexadecimal code. Enter this hexadecimal code in characteristic CC03. (Make sure that you are still logged in).

Upload the ConnectedCar code onto the Arduino and wire up the electronics.

The board is now ready to serve as the GateBox.

Interesting facts

If you press get notifications after entering a password for login (in CC02) or a password for change (in CC03), you will see that a notification of 0000 is received. This is done in order to erase the password value on the characteristic. The Windows Smartphone application will do this automatically.

Light states: writing a value of 1 means OFF, writing a value of 2 means ON.

Garage door states: writing a value of 1 means STOP, writing a value of 2 means OPEN and writing a value of 3 means CLOSE.

In the case of light (CC06) and garage (CC07) states, if you press notification, the updated value of the garage door / light state appears. This is used by the Windows Smartphone Application as a confirmation as to what is the current state of the garage door / lights. When the user first logs in, the current value of the garage door and light are waiting as notifications to be sent to the smartphone. Remember: the states of garage door and light can change from the GateBox panel and hence the smartphone app must be updated on the latest states of the light and garage door every time it logs into the GateBox panel.

Smartphone Application

There is a Windows App included into the folder "Windows Phone Application" that can be used. Currently, it is not on the Windows Store. Hence, it must be complied in visual studio and uploaded into a Windows Phone. It will have to be a Windows Phone and not an emulator. Overwhelming majority of phones today come with Bluetooth LE. However, it would be nice just to make sure that the phone of your choosing is compatible.

Anyone who follows the documentation for making Bluetooth Low Energy apps on Windows/Android/iPhone/Blackberry, you can easily make your own version of the app for this project. It would be nice to have different versions of the app available on their respective application markets and stores from where users can download these apps ready to use in this project.

Mirroring the Smartphone Application

For the purposes of demonstration, an e3iO Car PC was used http://e3io.com/2din-car-pc-small-edition. This Car PC runs Windows 8 – it is basically a PC in your car. Any number of software applications was used to mirror the smartphone's screen onto the Car PC. They can be downloaded from the internet for free. One such application is Car Dash (https://www.microsoft.com/en-us/store/apps/car-dash/9wzdncrfj0b9).

Getting an application into an auto manufacturer's dashboard requires overcoming a lot of regulations and cost a lot of money for testing. Services that make it possible for smartphone applications to be mirrored into the car's dashboard are:

1. AndroidAuto by Google
2. Apple CarPlay by Apple
3. MirrorLink by Car Connectivity Consortium
4. AppLink by Ford

An app that has undergone testing for one of the services above will only run on that service. Which means that if you want to run your application on AppLink and MirrorLink, you will have to pay the testing fee for AppLink and separate testing fee for MirrorLink. This investment would be preferable to someone interested in developing this further and manufacturing it into a commercial product.