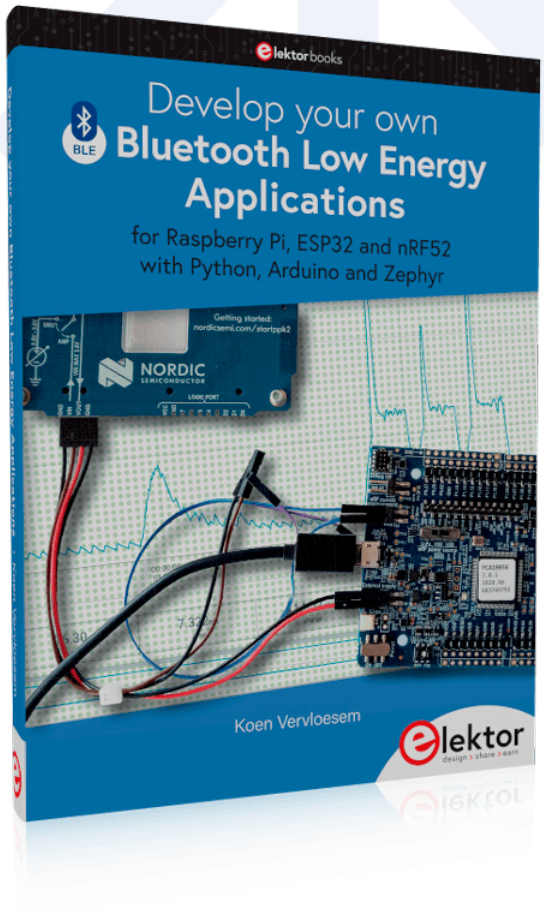
rétro-ingénierie d'un badge LED **Bluetooth Low Energy**

comment commander un appareil BLE avec un script Python

Koen Vervloesem (Belgique)

De nombreux appareils BLE (Bluetooth Low Energy) sont livrés avec leurs propres applications mobiles qui implémentent généralement un protocole personnalisé destiné à l'appareil et l'application correspondante, permettant à l'utilisateur de les commander. Souvent, il n'existe pas de spécification que vous pouvez lire pour créer votre propre application. La bonne nouvelle est que la rétro conception d'un appareil BLE est possible, permettant de le désassembler et de l'utiliser avec votre propre logiciel. Lisez la suite pour en savoir plus.



Note de la rédaction. *Cet article est un extrait du livre de 257 pages « Develop your own Bluetooth Low Energy Applications » (Elektor, 2022). L'extrait a été légèrement modifié par l'auteur pour correspondre aux normes éditoriales du magazine Elektor et pour constituer un projet complet et reproductible.*

Dans cet article, je vais rétro concevoir un badge LED BLE comme exemple de cette tâche complexe et très instructive d'ingénierie inverse. Rejoignez-moi ; nous découvrons le fonctionnement de l'appareil et examinons ses services et ses caractéristiques BLE. Je vais décompiler l'application mobile associée et analyser le trafic BLE entre l'application et l'appareil. Mon objectif est de créer un script Python pour contrôler le badge LED afin de se passer de l'application mobile officielle.

Étude du badge LED

Le badge LED d'AliExpress illustré à la **figure 1** est doté d'un écran LED de 11 x 55 pixels disponible en plusieurs couleurs. Il est compatible avec Bluetooth, mais sa version n'est pas précisée.

En scannant le code QR à l'arrière du badge, j'ai obtenu une erreur « HTTP 404 ». Sur Google Play, j'ai trouvé une application appelée

« *Bluetooth LED Name Badge* », développée par Shenzhen Lesun Electronics Co., Ltd (**figure 2**). Cette application permet d'envoyer du texte et des icônes vers le badge et dispose de quelques paramètres, y compris la vitesse de défilement.

Allumons le badge LED et utilisons l'application d'analyse BLE *« nRF Connect for Mobile »* [1] pour voir ce que l'appareil donne lorsqu'on le connecte. Appuyez deux fois sur le bouton inférieur. L'écran affiche alors une icône BLE. Vous remarquerez que l'application *nRF Connect* affiche des données spécifiques au fabricant et deux services BLE personnalisés : 0xfee7 et 0xfee0 (**figure 3**). Le nom de l'appareil est : LSLED. La connexion à celui-ci révèle ses services ainsi que les caractéristiques répertoriées dans le **tableau 1**.

Tableau 1. Caractéristiques du badge BLE

Service	Caractéristique	Propriété
0xfee7	0xfec7	ÉCRIRE
0xfee7	0xfec8	INDIQUER
0xfee7	0xfec9	LIRE
0xfee0	0xfee1	NOTIFIER, LIRE, ÉCRIRE

La lecture de la caractéristique 0xfec9 dans *nRF Connect for Mobile* renvoie la même valeur que les données spécifiques au fabricant. La lecture de 0xfee1 ne renvoie aucune donnée, et l'abonnement à ses notifications ne retourne rien non plus. La caractéristique *User Description* (description de l'utilisateur) de 0xfee1 est « Data ». Cette méthode ne donne rien d'utile.

Décompilation de l'application mobile

Puisque l'application du badge LED Bluetooth pour Android est évidemment capable de communiquer avec le badge LED, essayons de découvrir comment elle fonctionne réellement. Pour ce faire, nous allons la décompiler et fouiller dans les détails de son code source.

Téléchargez le fichier APK de l'application Android en utilisant un site de téléchargement d'APK tiers comme [2]. Il suffit de coller l'URL de l'application à partir du Google Play Store [3] dans le champ de recherche d'APKPure. Ensuite, vous pourrez télécharger le fichier. Ce fichier APK contient le code à octets Dalvik exécutable par Android. Pour comprendre ce que fait l'application, nous avons besoin de son code source. Bien que le développeur ait compilé son code source en Dalvik, nous pouvons le décompiler avec un décompilateur tel que jadx [4], qui convertit le code à octets Dalvik en code source Java.

Sous la version la plus récente de Windows [5], nous pouvons lancer l'interface graphique en double-cliquant sur le fichier **jadx-gui** dans le répertoire **bin**. Sous les autres systèmes d'exploitation (Linux ; MacOS), nous pouvons lancer l'interface graphique en exécutant bin/jadx-gui depuis la ligne de commande

Ouvrez le fichier APK téléchargé. Le programme décompile maintenant l'application et affiche une arborescence de ses paquets et fichiers Java à gauche. À ce stade-là, la recherche commence. Nous avons déjà

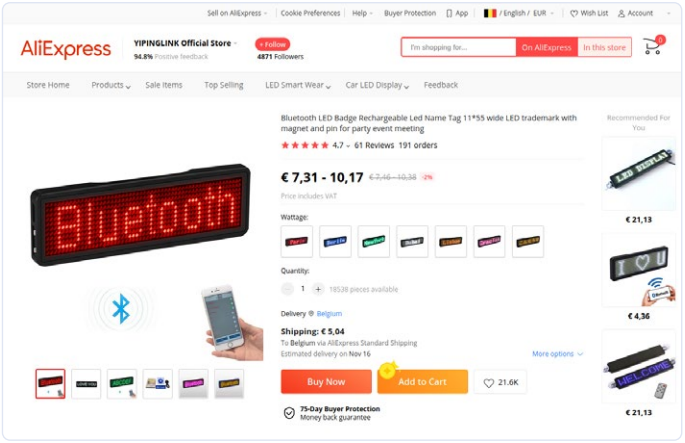


Figure 1. Ce badge LED Bluetooth semble être un appareil intéressant à rétroconcevoir !

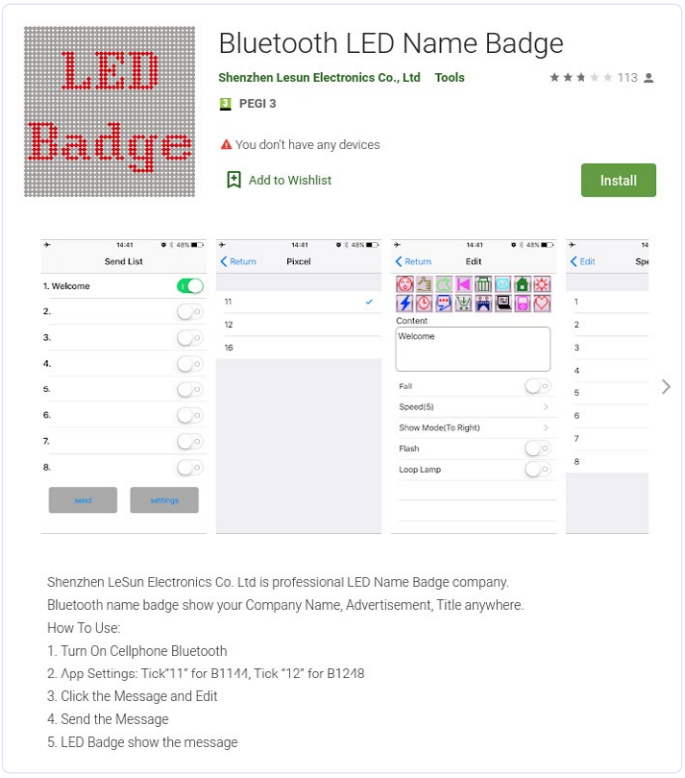


Figure 2. L'application « Bluetooth LED Name Badge app » dans le Google Play Store peut envoyer des commandes au badge LED.

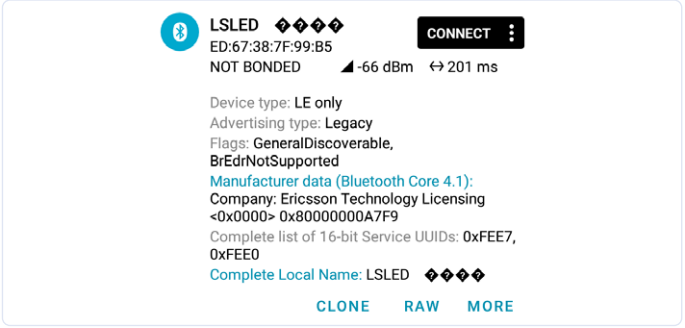


Figure 3. Le badge LED Bluetooth tel que détecté par nRF Connect for Mobile.

Listage 1.

```
public static final String UUID_CHARACTERISTICS_WRITE = "fee1";
public static final String UUID_SERVICE = "fee0";
```

Listage 2.

```
public static byte[] get64(List<SendContent> list, int i) {
    Iterator<SendContent> it = list.iterator();
    while (it.hasNext()) {
        Log.d("abcdef", "get64------------SendContent:" + it.next().toString());
    }
    byte[] bArr = new byte[64];
    bArr[0] = 119;
    bArr[1] = 97;
    bArr[2] = 110;
    bArr[3] = 103;
    bArr[4] = 0;
    bArr[5] = 0;
    bArr[6] = getFlash(list);
    bArr[7] = getMarquee(list);
    byte[] modeAndSpeed = getModeAndSpeed(list);
    for (int i2 = 0; i2 < 8; i2++) {
        bArr[i2 + 8] = modeAndSpeed[i2];
    }
    byte[] msgLength = getMsgLength(list, i);
    for (int i3 = 0; i3 < 16; i3++) {
        bArr[i3 + 16] = msgLength[i3];
    }
    bArr[32] = 0;
    bArr[33] = 0;
    bArr[34] = 0;
    bArr[35] = 0;
    bArr[36] = 0;
    bArr[37] = 0;
    byte[] date = getDate();
    for (int i4 = 0; i4 < 6; i4++) {
        bArr[i4 + 38] = date[i4];
    }
    for (int i5 = 0; i5 < 19; i5++) {
        bArr[i5 + 44] = 0;
    }
    bArr[63] = 0;
    return bArr;
}
```

quelques indices : les UUIDs des services et les caractéristiques trouvés dans *nRF Connect for Mobile*. Dans le menu **Navigation / Text search** (Navigation/Recherche de texte), vous pouvez entrer quelques termes de recherche.

Ceci est plus facile à dire qu'à faire. Le code ne mentionne pas le service 0xfee7 ni ses caractéristiques ! Il fait référence à l'autre service et à ses caractéristiques dans la classe `com.yannis.ledcard.ble.BleDevice`. Les éléments pertinents sont indiqués dans le **listage 1**.

Dans le code, vous pouvez maintenant rechercher ces deux constantes :
`UUID_CHARACTERISTICS_WRITE`
`UUID_SERVICE`
La façon la plus simple de le faire est probablement de cliquer avec le bouton droit de la souris sur le nom de la constante, puis de sélectionner **Find usage**. Cliquez ensuite sur l'une des instances trouvées pour ouvrir le fichier correspondant à cet emplacement.

En cherchant d'autres indices dans le code, je suis tombé sur un code intéressant qui traite les images et les modes d'affichage dans la classe `com.yannis.ledcard.util.LedDataUtil`.

Plus précisément, j'ai trouvé cette méthode Java donnée dans le **listage 2** qui crée une sorte d'en-tête pour les données.

Elle affiche un en-tête fixe (6 octets), un mode et une vitesse, une longueur de message et une date. Elle est appelée par `tSendHeader()` dans la classe mentionnée.

Procéder à l'ingénierie inverse en se plongeant dans le code et en analysant le fonctionnement de l'application n'était pas suffisant. J'ai donc commencé à utiliser l'application avec le badge LED tout en surveillant le trafic entre les deux appareils. Pour mieux comprendre, j'ai pu combiner les résultats obtenus à partir du code source de l'application avec le trafic en temps réel.

Renifler le trafic BLE entre le badge et l'application

Wireshark [6] renifle le trafic Bluetooth de votre téléphone en temps réel avec Android Debug Bridge. Connectez votre téléphone à votre ordinateur avec un câble USB et autorisez la connexion de débogage sur votre téléphone. Lancez Wireshark. Ce dernier devrait afficher **Android Bluetooth Btsnoop Net** comme l'une des interfaces disponibles. Double-cliquez sur l'interface et vous verrez défiler le trafic Bluetooth en direct ! Une meilleure solution consiste à utiliser le renifleur nRF pour le plugin BLE de Wireshark [7] avec un dongle nRF52840 comme renifleur.

Sur le badge BLE, appuyez deux fois sur le bouton inférieur jusqu'à ce qu'il affiche l'icône Bluetooth. Ensuite, sélectionnez le périphérique dans Wireshark pour afficher uniquement les paquets à destination et en provenance de ce périphérique spécifique.

Installez « Bluetooth LED Name Badge » sur votre téléphone Android et lancez l'application. Dans Wireshark, vous voyez que votre téléphone fait une demande de recherche et que le badge BLE affiche le nom de son appareil comme réponse. Sélectionnez le type de dispositif dans l'application. Pour le badge BLE de 11 x 55 pixels, le type est 11. Appuyez sur **yes** (oui).

L'application affiche une liste de messages à envoyer. La configuration par défaut est d'envoyer un message : **Welcome** (Bienvenue). Appuyez sur **Send** (Envoyer). L'application se connecte alors au badge LED et affiche le message de bienvenue sur son écran (**figure 4**).

Figure 4. Le badge LED Bluetooth affiche un message de bienvenue.

Dans Wireshark, nous voyons un paquet CONNECT_REQ suivi d'une demande d'attributs. Viennent ensuite quelques demandes d'écriture et de réponses. Pour obtenir une vision plus claire, cliquez avec le bouton droit de la souris sur *Write Request opcode* dans la section *Bluetooth Attribute Protocol* des détails du paquet, puis choisissez **Apply as Filter / Selected**. Vous pouvez également saisir : btatt.opcode == 0x12 pour filtrer l'affichage.

Dans ce cas, l'application envoie neuf paquets de demande d'écriture, chacun contenant 16 octets de données, comme indiqué ci-dessous :

```
77 61 6e 67 00 00 00 00 30 30 30 30 30 30 30 30
00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 e5 0a 18 0c 37 25 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 c6 c6 c6 c6 d6 fe ee c6 82 00 00 00 00 00 7c
c6 fe c0 c6 7c 00 00 38 18 18 18 18 18 18 18 3c
00 00 00 00 00 7c c6 c0 c0 c6 7c 00 00 00 00 00
7c c6 c6 c6 c6 7c 00 00 00 00 00 ec fe d6 d6 d6
c6 00 00 00 00 00 7c c6 fe c0 c6 7c 00 00 00 00
```

Si vous vérifiez la méthode Java `get64()` de la section précédente, vous reconnaîtrez les parties de l'en-tête : 77 61 6e 67 (équivalent hexadécimal du décimal 119 97 110 103). Ensuite, on obtient 00 deux fois, et encore 00 deux fois, qui sont respectivement les valeurs pour `getFlash()` et `getMarquee()`. Les huit octets suivants sont tous 0x30, qui représentent le mode et la vitesse. Les 16 octets suivants représentent la longueur du message. Puisque l'application affiche une liste de huit messages, cela pourrait être la liste des longueurs de ces messages. Les deux premiers octets sont 00 07, ce qui correspond au nombre de caractères du texte **Welcome**. On peut supposer que les 14 octets suivants représentent les longueurs des sept messages suivants (aucun, dans ce cas).

Ensuite, il y a six 00, et les six octets suivants représentent la date, qui est e5 0a 18 0c 37 25, dans ce cas. Converti en décimal, ça fait 229 10 24 12 55 37. J'ai exécuté cette application le 24 octobre 2021, à 12 h 55 min 37 s. Le mois, le jour et l'heure étaient corrects. L'en-tête se termine par 20 00.

Après cette étape, cinq paquets de 16 octets représentent en quelque sorte les sept caractères du texte **Welcome**. Découvrons comment cela se passe, en commençant par un message plus simple. Ouvrez de nouveau l'application et cliquez sur le premier message. Remplacez le texte par **W** et activez **Flash**. Revenez à l'écran principal, cliquez sur le deuxième message et ajoutez le texte : **e**. Pour ce deuxième message, réglez la vitesse sur 8, le mode sur **Right** et activez **Marquee**. Retournez à l'écran principal. Activez ensuite le curseur à côté du deuxième message et cliquez sur **Send** (envoyer) tout en observant les paquets du *Bluetooth Attribute Protocol* dans Wireshark.

Là, le badge LED affiche la lettre *W* qui se déplace vers la gauche et clignote. Ensuite, il affiche la lettre *e* se déplaçant vers la droite à double vitesse et un cadre de pixels mobiles aux alentours. Dans ce cas, l'application a envoyé six paquets de demande d'écriture, chacun contenant 16 octets de données – voir ci-dessous :

```
77 61 6e 67 00 00 01 02 30 71 30 30 30 30 30 30
00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 e5 0a 18 10 06 13 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 c6 c6 c6 c6 d6 fe ee c6 82 00 00 00 00 00 7c
c6 fe c0 c6 7c 00 00 00 00 00 00 00 00 00 00 00
```

Le premier paquet commence avec les mêmes 6 octets, mais suivi de la valeur de `getFlash()` qui est 01 et celle de `getMarquee()` qui est 02. C'est parce que nous avons activé **Flash** pour le premier message et **Marquee** pour le deuxième message.

L'octet suivant est toujours 0x30, car nous n'avons rien changé quant au mode et à la vitesse du premier message. Mais l'octet suivant est désormais 0x71. Nous avons changé la vitesse à 8 et le mode à **Right** (droite). Donc, le quartet de gauche de cet octet code apparemment, la vitesse (la vitesse 4 est codée par 3 et la vitesse 8 par 7) et le quartet de droite code le mode. Le deuxième paquet indique 00 01 comme longueur du premier message et la même valeur pour la longueur du deuxième message. Cela confirme notre hypothèse. Ensuite, la date est codée.

Il reste maintenant deux paquets. Convertissez les valeurs hexadécimales en binaire et formatez-les octet par octet et en séquence, puis divisez-les en onze lignes (oui, parce que l'écran fait 11 pixels de haut) comme suit :

```
00000000
11000110
11000110
11000110
11000110
11010110
11111110
11101110
11000110
10000010
00000000

00000000
00000000
00000000
00000000
01111100
11000110
11111110
11000000
11000110
01111100
00000000
```

Listage 3.

```
"""Find BLE LED badges.
Copyright (c) 2022 Koen Vervloesem
SPDX-License-Identifier: MIT
"""
import asyncio
from bleak import BleakScanner
num_devices = 0
def device_found(device, advertisement_data):
    """Show device details if it's a BLE LED badge."""
    global num_devices
    if device.name.startswith("LSLED"):
        num_devices += 1
        print(
            f" () - RSSI: "
        )
async def main():
    """Scan for BLE devices."""
    print("Searching for LED badges...")
    scanner = BleakScanner()
    scanner.register_detection_callback(device_found)
    await scanner.start()
    await asyncio.sleep(5.0)
    await scanner.stop()
    if not num_devices:
        print("No devices found")
if __name__ == "__main__":
    asyncio.run(main())
```

Vers la fin, des 0 de remplissage sont utilisés pour remplir un bloc de 16 octets (non représentés ici). Si vous appliquez le même décodage au premier message avec **Welcome** et que vous placez les bitmaps des lettres les uns à côté des autres, vous obtiendrez un 1 pour chaque pixel activé et un 0 pour chaque pixel désactivé, comme illustré ici :

```
00000000000000000000000000000000000000000000000000000000000000000
11000111000000000001110000000000000000000000000000000000000000000
11000111000000000001100000000000000000000000000000000000000000000
11000111000000000001100000000000000000000000000000000000000000000
11000111001111100001100001111100011111001110011000111110000000000
11010111011000110000110001100011011000110111111110110001100000000
11111111011111110000110001100000011000110110101101111111000000000
11101111011000000000110001100000011000110110101101100000000000000
11000111011000110000110001100011011000110110101101100011000000000
10000010011111100001111000111110001111100110001100111110000000000
00000000000000000000000000000000000000000000000000000000000000000
```

Vous avez maintenant une bonne idée du format des données à envoyer au badge LED afin d'afficher du texte sur l'écran. Rassemblons tout cela et créons un script Python pour envoyer vos propres images au badge LED.

Création d'images aléatoires sur le badge LED avec *Bleak*

Le badge LED à Bluetooth vous permet également d'envoyer des petits bitmaps 11 x 11 prédéfinis. Mais maintenant que vous connaissez le format de données à envoyer, vous pouvez également choisir un bitmap arbitraire, par exemple, pour couvrir la totalité de l'écran 11 x 55. Faisons cela avec *Bleak* [8], une bibliothèque Python multiplateforme pour BLE. Saisissez :

```
pip3 install bleak
```

Créons d'abord un script qui recherche les badges BLE en fonction de leur nom pendant cinq secondes, puis affiche leur adresse Bluetooth et leur RSSI – voir **listage 3**.

Si vous l'exécutez avec deux badges LED qui sont tous deux en mode **Listen**, le script affiche :

```
$ python3 find_led_badge.py
Searching for LED badges...
ED:67:38:80:0D:E2 (LSLED) - RSSI: -74
ED:67:38:7F:99:B5 (LSLED) - RSSI: -83
```

Exploitons maintenant les connaissances acquises dans la section précédente. Examinons ce script qui envoie les commandes appropriées au badge LED pour afficher une image – voir **listage 4**.

Outre *Bleak*, ce script utilise également la bibliothèque *Pillow* [9] :

```
pip3 install Pillow
```

Les quatre premières commandes BLE sont codées en dur. Dans la première commande, 0x34 définit le mode et la vitesse du premier message. 4 signifie un mode statique : le message est affiché sous forme d'image fixe. Dans la deuxième commande, 0x07 définit la longueur du message et le nombre de caractères de 8 pixels. L'écran a une largeur de 55 pixels, et 7 x 8 = 56, nous devons donc coder l'image en 7 caractères. Les deux commandes suivantes devraient encoder la date actuelle, mais ce n'est pas vraiment nécessaire. Il suffit d'envoyer seize 0 pour chacune de ces commandes.

`chunks()` est une fonction auxiliaire pour diviser un tableau d'octets en une liste de tableaux, chacun de 16 octets. Elle est utilisée dans la fonction `commands_for_image()` qui convertit un fichier image en octets représentant des caractères sur l'écran. À la fin de la fonction, ces octets sont divisés en regroupement de 16 octets.

`commands_for_image()` ouvre une image avec *Pillow* et charge ses pixels. Pour chacun des sept caractères de l'écran, elle code le caractère sous forme de 11 octets : un pour chaque ligne. Tous ces octets sont ajoutés à `octets_image`.

À la fin de l'image, le tableau d'octets est complété par des octets supplémentaires afin d'obtenir un multiple de 16 octets. Et, finalement, il est divisé en bloc de 16 octets.

La fonction `main()` concrétise tout cela. Elle se connecte au périphérique et crée une liste de commandes : les quatre commandes de l'en-tête sont étendues avec les commandes pour l'image. Ensuite, chacune de ces commandes est écrite dans la caractéristique avec l'UUID 0xfee1. Le code principal en bas de page vérifie si vous avez

Listage 4.

```
"""Display an image on a BLE LED badge.
Copyright (c) 2022 Koen Vervloesem
SPDX-License-Identifier: MIT
"""
import asyncio
import sys
from PIL import Image
import bleak
WRITE_CHAR_UUID = "0000fee1-0000-1000-8000-00805f9b34fb"
COMMAND1 = bytes([0x77, 0x61, 0x6E, 0x67, 0x00, 0x00, 0x00, 0x00, 0x34, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30])
COMMAND2 = bytes([0x00, 0x07, *(14 * [0x00])])
COMMAND3 = bytes(16 * [0x00])
COMMAND4 = bytes(16 * [0x00])
def chunks(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i : i + n]
def commands_for_image(image):
    """Return commands to show an image on the BLE LED badge."""
    image_bytes = bytearray()
    with Image.open(image) as im:
        px = im.load()
        for i in range(7):  # 7x8 = 56 -> 7 bytes next to each other
            for row in range(11):
                row_byte = 0
                for column in range(8):
                    try:
                        pixel = int(
                            px[(i * 8) + column, row][3] / 255
                        )
                    except IndexError:
                        pass  # Ignore the 56th pixel in a full row
                    row_byte = row_byte | (pixel << (7 - column))
                image_bytes.append(row_byte)
    # Fill the end with zeroes to have a multiple of 16 bytes
    image_bytes.extend(bytes(16 - len(image_bytes) % 16))
    # Split image bytes into 16-byte chunks
    return list(chunks(image_bytes, 16))
async def main(address, filename):
    """Connect to BLE LED badge and send commands to show an image."""
    try:
        async with bleak.BleakClient(address) as client:
            commands = [COMMAND1, COMMAND2, COMMAND3, COMMAND4]
            commands.extend(commands_for_image(filename))
            for command in commands:
                await client.write_gatt_char(WRITE_CHAR_UUID, command)
    except asyncio.exceptions.TimeoutError:
        print(f"Can't connect to device .")
    except bleak.exc.BleakError as e:
        print(f"Can't write to device : ")
if __name__ == "__main__":
    if len(sys.argv) == 3:
        address = sys.argv[1]
        filename = sys.argv[2]
        asyncio.run(main(address, filename))
    else:
        print(
            "Please specify the BLE MAC address and image filename."
        )
```

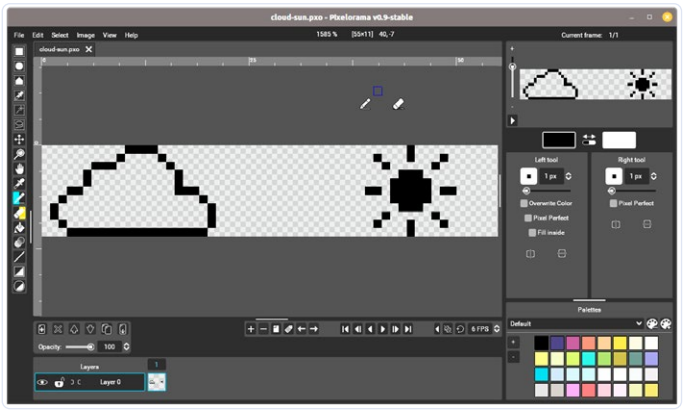


Figure 5. Pixelorama est un éditeur d'images 2D à code source ouvert.

Figure 6. En envoyant les commandes BLE appropriées, vous pouvez afficher vos propres images de 11 x 55 pixels sur le badge LED.

fourni deux attributs sur la ligne de commande. Le premier est attribué à l'adresse Bluetooth, et le second, au nom du fichier.

Avant d'exécuter ce code, vous devez préparer une image d'exactement 11 x 55 pixels, avec un éditeur de pixels tel que Pixelorama [10]. Définissez un pixel pour chaque LED que vous souhaitez allumer sur l'écran. La **figure 5** montre un exemple d'une image de nuages et de soleil que j'ai créée dans Pixelorama.

Exportez l'image sous forme de fichier .PNG. Ensuite, mettez le badge LED en mode **Discovery** en appuyant deux fois sur le bouton du bas jusqu'à ce que l'icône Bluetooth apparaisse. Exécutez le script Python avec deux paramètres : l'adresse Bluetooth du badge et le nom de fichier de l'image :

```
$ python3 display_led_badge.py ED:67:38:7F:99:B5 cloud-sun.png
```

Conclusion

Le badge BLE devrait afficher votre image (**figure 6**). Voilà, vous avez réussi à rétro concevoir le badge BLE et vous pouvez dès maintenant l'utiliser dans votre propre code. Bien entendu, il est possible d'améliorer le script que je vous ai proposé, notamment pour le rendre plus convivial. J'espère que cela vous incitera à appliquer le même principe de rétro-ingénierie à d'autres appareils BLE.

220439-04 — VF : Asma Adhimi

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (koen@vervloesem.eu) ou contactez Elektor (redaction@elektor.fr).

LIENS

[1] L'application nRF Connect for Mobile : https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-mobile
[2] Fichier APK : https://apkpure.com
[3] L'application sur Google Play Store : https://play.google.com/store/apps/details?id=com.yannis.ledcard
[4] Le décompilateur jadx : https://github.com/skylot/jadx
[5] Dernière version du décompilateur jadx : https://github.com/skylot/jadx/releases
[6] Wireshark : https://www.wireshark.org
[7] nRF Sniffer for Bluetooth LE : https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-bluetooth-le
[8] Bleak : https://bleak.readthedocs.io
[9] La bibliothèque Pillow : https://pillow.readthedocs.io
[10] Pixelorama : https://orama-interactive.itch.io/pixelorama
[11] Page de ressources/informations sur le livre : https://www.elektor.fr/develop-your-own-bluetooth-low-energy-applications
[12] Le code sur GitHub : https://github.com/koenvervloesem/bluetooth-low-energy-applications