Object

This document presents the integration manual of LibCardio.

Scope of application

This document applies only to the medical device LibCardio.

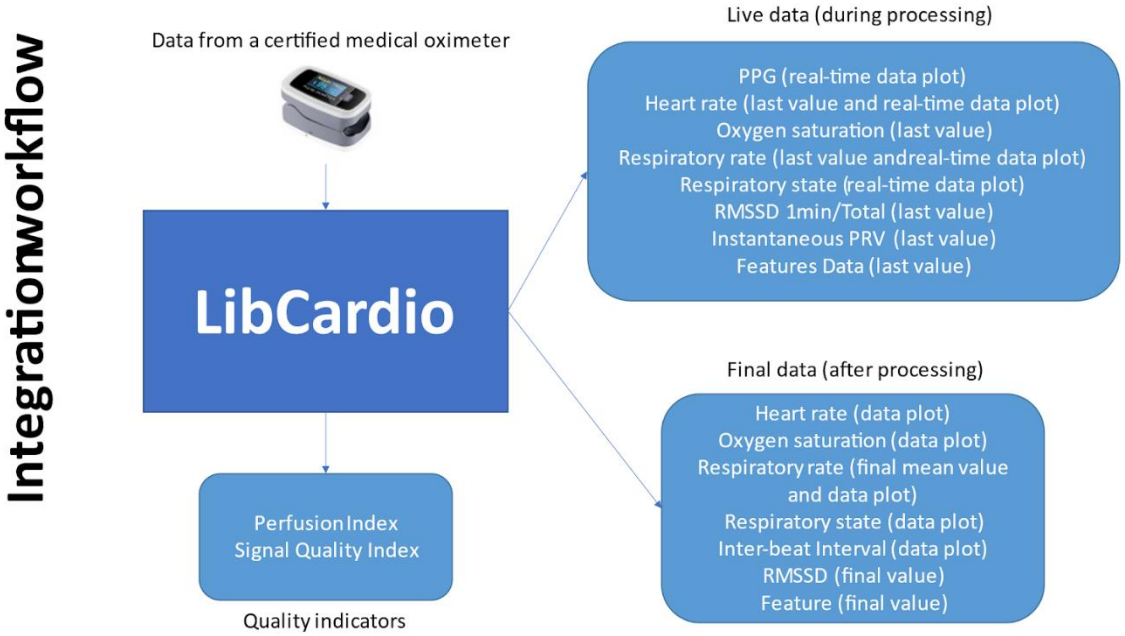
Revision history

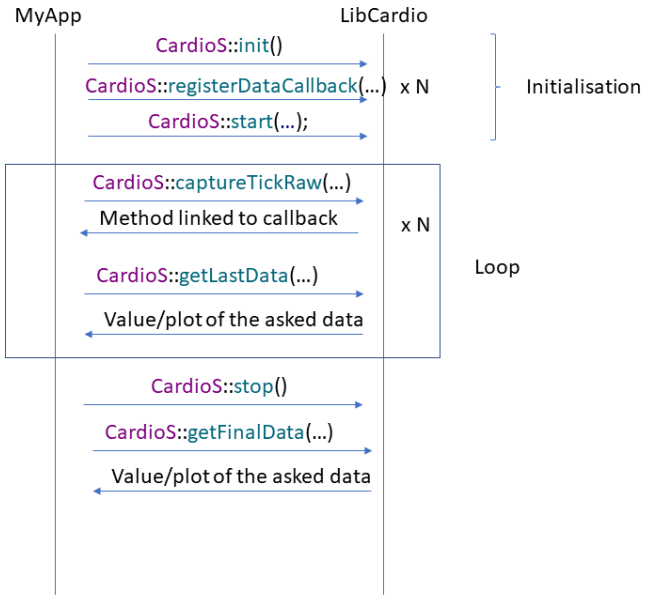
Version	Date	Changes in the record	Editor
01	22/03/2023	Creation	Mélaine GAUTIER
02	20/06/2023	Ajout des informations de label et de fréquence d'acquisition	Mélaine GAUTIER

Table des matières

1. Description du dispositif médical LibCardio

Ce projet, avec sa bibliothèque logicielle LibCardio, vise à prévenir de manière significative les risques de maladie cardio-respiratoire en apportant des informations les plus significatives disponibles.





Ce manuel présentera d'abord le fonctionnement d'un oxymètre qualifié, puis l'implémentation en C ++, Java et pour application mobile.

Comme la bibliothèque est développée en C ++, il est recommandé de se référer au guide C ++ lors de l'écriture du « Bridge » vers d'autres plateformes.

2. Description du produit

2.1. Principe de fonctionnement

2.1.1. Objectif du produit

L'objectif du LibCardio est de fournir une bibliothèque apportant des outils simples, rapides et fiables à intégrer dans une application offrant aux médecins généralistes et aux professionnels de santé pour détecter les premiers signes de risques de maladies cardio-respiratoires pour tous les patients.

2.1.2. Comment fonctionne le produit ?

La bibliothèque LibCardio dispose d'élément d'entrée et de sortie.

Les éléments d'entrée de la bibliothèque sont entre autres :

- Les données brutes : les fondamentales de ces données sont : le signal photopléthysmographie (PPG), le Heart rate (HR), la Saturation pulsée en oxygène (SpO_2), Inter-Beat Interval (IBI). Ces données sont acceptées par la bibliothèque à condition qu'elles respectent les configurations requises (format de données…) par cette dernière ;
- Les données en provenance d'un dispositif certifié médical (comme un oxymètre, une montre connectée…), testé et validé afin d'assuré l'exactitude des résultats fournis par la bibliothèque. La connexion avec la bibliothèque peut s'effectuer par envoi de paquets de données au format requit ou un dispositif Bluetooth ou USB reconnu par cette dernière. Le dispositif d'entrée utilisé actuellement est un Oxymètre du fournisseur Berry utilisant un protocole privé ;
- Un fichier généré en interne disposant déjà les données et les configurations requises par la bibliothèque.

Les données reçues par la bibliothèque sont ensuite traitées au fur et à mesure qu'elles sont acquises et les différents algorithmes sont appliqués afin de générer les données de sorties (en instantané et à la fin de l'enregistrement) parmi lesquelles en plus des données d'entrées, on a : la qualité du signal, la RMSSD, l'amplitude cardiaque, la variabilité cardiaque, la fréquence respiratoire.

La bibliothèque LibCardio est destinée à être utilisé en association avec une autre application (middleware) qui est considérée comme étant l'interface utilisateur. C'est dans cette application que les données de sorties de la bibliothèque vont être interprétées en fonction du besoin. Ainsi, on pourra avoir des informations comme la rigidité cardiaque, l'élasticité des artères, la tension artérielle, le rythme respiratoire…

2.2. Télécharger la bibliothèque

La bibliothèque est téléchargeable via le lien et le mot de passe fournis lors de l'achat ou de la mise à jour. Vous récupérerez un zip qu'il faudra extraire.

Il contiendra le fichier « .h » (header) et la bibliothèque pour la version que vous utilisez (linux, Windows, mac, iOS, Android, …) ainsi qu'un exemple selon le langage d'intégration.

Si vous êtes encore en abonnement maintenance, un mail contenant un nouveau lien, les informations de mises à jour/évolutions vous sera transmis lors de chaque nouvelle version de la bibliothèque.

Pour connaître la version de la bibliothèque utilisée, le chapitre 5.2.5 et le chapitre 5.3.6 présente une méthode/fonction permettant de connaître la version de la bibliothèque courante.

2.3. Oxymètre

Les oxymètres reconnus par la bibliothèque sont les oxymètres de la compagnie Berry qui intègrent le protocole propriétaire SENSORIA ANALYTICS.

Les autres oxymètres peuvent être utilisés en envoyant directement des données brutes selon le protocole défini par SENSORIA ANALYTICS.

Ils peuvent être connecté par 2 moyens :

2.3.1. via câble USB :

Le BM3000B est un récent oxymètre d'impulsion pour mesurer la saturation en oxygène (SpO2) et le rythme cardiaque. Ce produit ressemble à un disque U, sans écran, sans batterie et sans boutons. Il a deux connecteurs, l'un pour la connexion avec la sonde SpO2, l'autre pour la connexion avec le PC. L'alimentation nécessaire est fournie par PC ou téléphone mobile connecté avec le produit. Il convient aux établissements de santé, à l'usage familial, ainsi qu'aux soins physiques lors de sports ou d'activités dans des environnements extrêmes. (Vous pouvez l'utiliser avant ou après le sport, mais il n'est pas recommandé de l'utiliser pendant le sport).

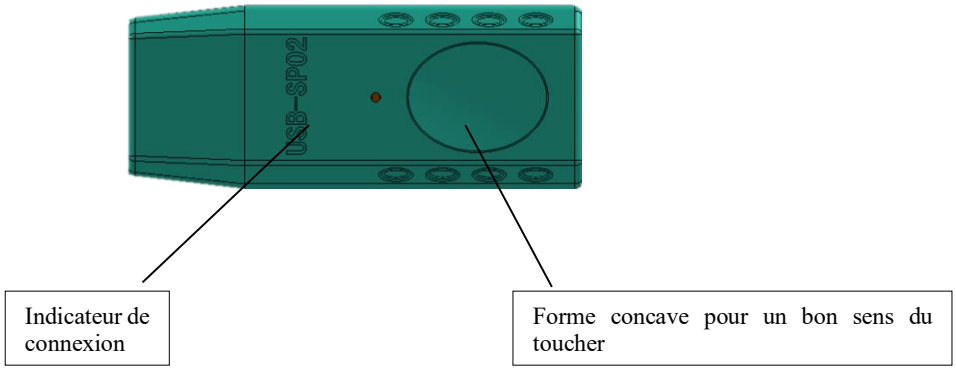
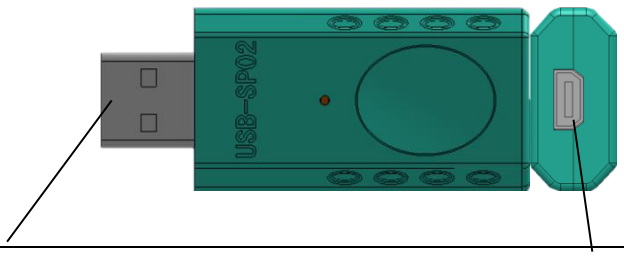


Figure 1



Connecteur de PC	Connecteur de sonde Spo_2

Figure 2

2.3.2. via Bluetooth

2.3.2.1. Présentation de l'affichage

2.3.2.2. Fonction de communication Bluetooth

L'oxymètre de pouls BM1000D est doté de la fonction de communication Bluetooth. Il peut envoyer des données à un terminal intelligent et à un ordinateur (le logiciel associé CARDIOSENSYS a été installé.) Avec une fonction de communication travers un DONGLE fourni avec l'oxymètre.

2.3.2.3. Description de la fonction

a. Lorsque les données ont été affichées à l'écran, appuyez brièvement une fois sur le bouton "POWER / FUNCTION" pour faire pivoter la direction d'affichage. (Comme le montre le **Figure 12**)

b. Appuyez ensuite à nouveau brièvement sur le bouton « POWER / FUNCTION » pour rétablir le sens d'affichage précédent. Et la sonnerie indiquant que disparaîtra en même temps, la sonnerie sera désactivée. (Comme le montre **la figure 13**)

c. Appuyez longuement sur le bouton « POWER / FUNCTION », l'indication Bluetooth disparaîtra et la fonction Bluetooth sera désactivée. (Comme le montre le **Figure 14**)

Remarque : Lorsque la fonction Bluetooth n'est pas connectée avec succès, l'indication Bluetooth clignote. Lorsque la fonction Bluetooth est connectée avec succès, l'indication Bluetooth continuera d'être allumée.

d. Lorsque le signal reçu est inadéquat, "  " s'affiche à l'écran. (Comme le montre La **Figure 15**)

e. Le produit sera automatiquement éteint lorsqu'aucun signal n'est déclenché après 10 secondes.

Figure 12

Figure 13

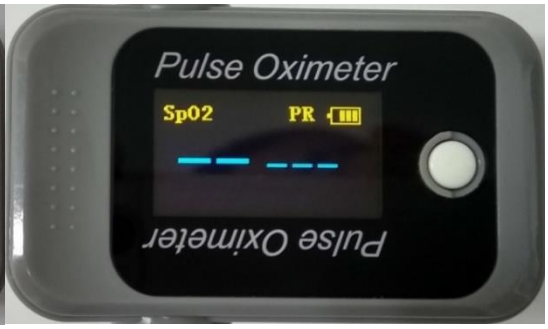


Figure 14 **Figure 15**

2.4. Mode d'emploi

1. Tenez le produit dans une main, le panneau avant face à la paume. Placez le gros doigt de l'autre main sur le signe de presse du couvercle du compartiment des batteries, appuyez vers le bas et ouvrez le couvercle en même temps. L'armoire batterie est ouverte comme le montre la **figure 16**.
2. Installez les piles dans les fentes en respectant les symboles «+» et «-», comme illustré à la **figure17**.
3. Appuyez sur la touche Clip de la figure 1 et ouvrez le clip. Laissez le doigt de la personne testée dans les coussinets en caoutchouc de la pince, assurez-vous que le doigt est dans la bonne position, comme illustré à **la figure 18**, puis clippez le doigt.
4. Appuyez sur le bouton d'alimentation et de fonction du panneau avant pour allumer le produit. Utiliser le premier doigt, le majeur ou l'annulaire pour faire le test. Ne raccrochez pas le doigt et gardez la personne testée à la caisse pendant le processus. Les lectures seront affichées sur l'écran OLED un moment plus tard, comme illustré à la **figure 19**.

Attention :

- Les électrodes positives et négatives des batteries doivent être installées correctement. Dans le cas contraire, l'appareil sera endommagé.
- Lors de l'installation ou du retrait des piles, veuillez suivre la séquence d'opérations appropriée. Sinon, le compartiment de la batterie sera endommagé.
- Si l'oxymètre d'impulsion n'est pas utilisé pendant longtemps, enlevez s'il vous plaît les piles de celui-ci.
- Assurez-vous de placer le produit sur le doigt dans une direction correcte. La partie LED du capteur doit être à l'arrière de la main du patient et la partie photodétecteur à l'intérieur. Assurez-vous d'insérer le doigt à la profondeur appropriée dans le capteur de sorte que l'ongle est juste en face de la lumière émise par le capteur.
- Ne secouez pas le doigt et ne laissez pas l'espèce testée pendant le processus.
- La période de mise à jour des données est inférieure à 30 secondes.

Figure 16

Figure 17

Figure 18

Figure 19

3. Utilisation avec un PC

1. Installer le logiciel de gestionnaire de données et le pilote de logiciel sur le PC. (Vous pouvez obtenir le logiciel et le pilote auprès de Sensoria Analytics)
2. Connectez l'appareil USB-SPO2 et la sonde SpO2 (ou le dongle Bluetooth dans le cas de l'utilisation de l'oxymètre Bluetooth), puis branchez-le sur le PC. (Comme le montre la figure 11)

Figure 11

3. Mettez le doigt dans la sonde SpO2 et vous pouvez lancer le test de vérification ou votre application.

4. Paramètre de configuration des oxymètres

4.1. Berry

4.1.1. Bluetooth Service Information (UUIDs)

Comm Service: 49535343-FE7D-4AE5-8FA9-9FAFD205E455

Send Characteristic: 49535343-1E4D-4BD9-BA61-23C647249616
Receive Characteristic: 49535343-8841-43F4-A8D4-ECBE34729BB3
Rename Characteristic: 00005343-0000-1000-8000-00805F9B34FB
MAC Address Characteristic: 00005344-0000-1000-8000-00805F9B34FB

NOTE: “Rename Characteristic” et “MAC Address Characteristic” ne sont pas supporté par le Bluetooth classique (e.g. BT 3.0 and below).

4.1.2. USB-ComPort Settings

Baudrate: 115200
Bits: 8
Stopbit: 1
Paritybit: none

4.1.3. Device Packet (Device to Host):

Longueur des packets: 20 bytes

Fréquence des paquets: Adjustable （Default is 100Hz but our recommendation is to set to 200Hz)

Packet Format:
Byte0: 0xff Head mark1
Byte1: 0xaa Head mark2
Byte2**:** PktIndex Packet index
…
Byte19: Checksum Checksum of the whole packet

PacketFreq: Current packets sending frequency

valid values: 1, 50, 100, 200, means 1Hz, 50Hz, 100Hz, 200Hz

Checksum Range (0~255)

Formula: Checksum = (Byte0+Byte1+…+Byte18) % 256, (% est l'opérateur modulo)

4.1.4. Host Command (Host to Device)

Command length: 1 byte
Command type:
0xf0 ---------- 50 Hz Package rate
0xf1 ---------- 100 Hz Package rate (Default)
0xf2 ---------- 200 Hz Package rate
0xf3 ---------- 1 Hz Package rate
0xf4 ---------- ADCSample is Original sampling waveform
0xf5 ---------- ADCSample is Filtered sampling waveform
0xf6 ---------- Stop sending packet
0xff ---------- Get Software Version
0xfe ---------- Get Hardware Version
0xfd ---------- Get Bluetooth Version (Optional)
0xfc ---------- Get Unique ID (Optional)

Par exemple:
Software Version Package Example (20 bytes):
0xff 0xaa 0x53 0x56 0x31 0x2e 0x30 0x34 0x2e 0x30 0x30 0x2e 0x33 0x36 0x00 0x00 0x00 0x00 0x00 0x3a
0xff 0xaa ------ Packet head mark
0x53 ------ ASCII 'S', mark of software version

0x56 0x31 0x2e 0x30 0x34 0x2e 0x30 0x30 0x2e 0x33 0x36 ------ ASCII string 'V1.04.00.36', current software version content
0x00 0x00 0x00 0x00 0x00 ------ Padding bytes of packet
0x3a ------ Checksum byte
Hardware Version Package Example:
0xff 0xaa 0x48 0x56 0x32 0x2e 0x30 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xd7
0xff 0xaa ------ Packet head mark
0x48 ------ ASCII 'H', mark of hardware version
0x56 0x32 0x2e 0x30 ------ ASCII string 'V2.0', current hardware version content
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 ------ Padding bytes of packet
0xd7 ------ Checksum byte
Unique ID Package Example: (Assumed that unique id is "**0x02020089541e1c17**")
0xff 0xaa 0x55 0x30 0x32 0x30 0x32 0x30 0x30 0x38 0x39 0x35 0x34 0x31 0x65 0x31 0x63 0x31 0x37 0x8e
0xff 0xaa ------ Packet head mark
0x55 ------ ASCII 'U', mark of unique id
0x30 0x32 0x30 0x32 0x30 0x30 0x38 0x39 0x35 0x34 0x31 0x65 0x31 0x63 0x31 0x37 ------ ASCII string '**02020089541e1c17**', unique id content

0x8e ------ Checksum byte

5. Intégration dans un projet C++

5.1. Prérequis

- Un compilateur C++
  - Pour windows: Cygwin ou MinGW 64 pour compiler du code C++

Les fréquences acceptées par la bibliothèque sont 50, 100 ou 200Hz (recommandé).

5.2. Exemple 1 : avec la passerelle BridgeCppLibCardio

C'est la manière la plus simple d'intégrer la bibliothèque avec des fonctions très généralistes

5.2.1. Etape 1 : Initialisation du processus

Tous les encadrés de cette section son extrait du fichier à inclure ***bridgeCppLibCardio.h***

Il faut tout d'abord fournir des informations générales si les valeurs par défaut ne vous conviennent pas.

```
enum ConfigItem{
  SRC=0, // source if not raw data
```

```
    PACKAGE_RATE, // int default 200
    CORE_RATE, // int default 500
    NB_CORE, // int min 1, max 8, default 4
};
void setConfiguration(const ConfigItem &item, const std::string &value);
std::string getConfiguration(const ConfigItem &item) const;

enum ProfileItem{
    GENDER=0, //'m','w' or 'o'
    AGE,    // age (int)
    HEIGHT, // height(int in cm)
    WEIGHT, // weight (int)
    SMOKERLEVEL, // smokerLevel('l','m' or 'h')
    PHYSICALACTIVITY  // sportLevel('l','m' or 'h')
};
void setPatientData(const ProfileItem &item, const std::string &value);
std::string getPatientData(const ProfileItem &item) const;
```

La configuration permet de préciser le chemin/source si vous utiliser un fichier ou un oxymètre connecté. Vous pouvez aussi au besoin modifier la fréquence des paquets (par défaut 200Hz) et la fréquence interne (fortement déconseillé). Enfin selon votre ordinateur, vous pouvez souhaiter utiliser plus ou moins de cœur pour votre application (par défaut, on utilise 4 cœurs).

Les informations du patient permettent d'affiner certains résultats en fonction des profils.

Les valeurs par défaut sont :

- Pour le genre 'o' (other)
- Pour le niveau de fumeur 'l' (low)
- Pour le niveau d'activité 'm' (medium)

Les autres informations n'ont pas de valeur par défaut et il est recommandé de les définir.

Une fois cette étape faite, on peut initialiser la bibliothèque.

```
enum DeviceType{
    BERRY=0,
    NONIN,
    FILE, // Read a file by following file time
    FILE_FAST, // Read a file but on one go
    RAW
};
```

```
enum Version{
   V1_4=0, // OBSOLETE
   V2_0, // OBSOLETE
   V2_02
};
// For for other than the raw connection - you need to specify setConfiguration(SRC, path/source);
bool init(DeviceType type, Version v=V2_02);
```

La fonction init va mettre la bibliothèque dans un état d'attente des données après avoir (re)initialisé les paramètres internes. **Cette méthode doit absolument être appelée avant d'appeler les fonctions qui vont être présentées ci-dessous.**

Elle doit aussi être rappelée après une modification des paramètres ou du profil patient ou avant une nouvelle mesure pour remettre tous les paramètres internes à zéro.

5.2.2. Etape 2 : Envoyer les données à la bibliothèque

```
bool tick();
bool tick(const std::vector<uint8_t>& rawData);
bool tick(const std::vector<std::string>& rawData, bool isDecimal=false);
```

La méthode tick(…) envoie les données à la bibliothèque :

- tick() sera utilisé avec tous les types de dispositif sauf les données brutes
- tick(…) permettent d'envoyer des données brutes au format hexadécimal ou décimal à la bibliothèque (selon la méthode utilisée).

Une fois que les données sont reçues par la bibliothèque, les valeurs temps réelles sont disponibles.

5.2.3. Etape 3 : Arrêter la bibliothèque

Il est obligatoire d'arrêter la bibliothèque avant de quitter l'application car sinon, elle continuera à tourner en tâche de fond et consommera de la mémoire inutilement.

De plus in certain nombre de résultats ne sont disponible qu'après l'appel de la fonction arrêt.

Pour arrêter le processus, il faut appeler la méthode :

```
bool stop();
```

5.2.4. Etape 4 : Récupérer les valeurs d'un signal

5.2.4.1. Signaux temps réels

Différentes fonctions permettent de récupérer de nombreux paramètres au cours de l'enregistrement d'une mesure :

- isFinger pour savoir si le doigt est bien détecté
- getLorenz permet d'avoir la variabilité instantanée du rythme du pouls.
- get(DataType) permet de récupérer les valeurs instantanées. Si une valeur n'est pas disponible -1 sera renvoyé. Voici la liste des éléments disponibles en temps réel :
    - SA_ARRHYTHMIA : indicateur d'une potentielle arythmie – valeur disponible après 30 secondes puis la valeur change toutes les 20 secondes
    - SA_BATTERY : si le dispositif nécessite des piles, l'état de charge est indiqué
    - SA_HR : rythme cardiaque – valeur disponible après 1 seconde environ, la valeur est mise à jour toutes les secondes
    - SA_PI : indicateur de perfusion (circulation générale du sang) - peut ne fournir une valeur qu'après quelques millisecondes
    - SA_RMSSD : calcul de la variabilité cardiaque – disponible juste après les premières valeurs de SA_HR, la valeur est mise à jour toutes les secondes
    - SA_RR : rythme respiratoire – valeur disponible après 30 secondes puis la valeur change toute les 3 secondes
    - SA_SPO2 : niveau de saturation en oxygène – valeur disponible après 1 seconde environ, la valeur est mise à jour toutes les secondes
    - SA_SQI : indicateur de qualité du signal – nécessite quelques secondes avant d'avoir la première valeur
    - SA_TIME : donne le temps courant
- getPoints(DataType) permet de récupérer une liste de points instantanées formaté selon le modèle suivant (temps, valeur). Si une valeur n'est pas disponible un vecteur vide sera renvoyé. Voici la liste des éléments disponible en temps réel :
    - SA_PPG_RAW : PPG signal brut de la source - peut ne fournir une valeur qu'après quelques millisecondes.
    - SA_PPG_FIL : PPG signal filtré) - peut ne fournir une valeur qu'après quelques millisecondes.

```
enum DataType{
    SA_PI=0,        /// Perfusion - Index value between 0 and 20 ///
    SA_SQI,         /// Signal Quality Index - value between 0 and 100 ///
    SA_HR,          /// Heart Rate - value between 0 and 220 - get & getPoints ///
    SA_SPO2,        /// Oxygen Saturation - value between 0 and 100 - get & getPoints ///
    SA_RMSSD,       /// Heart Variability of the total measurement - value between 0 and 400 ///
```

```
    SA_BATTERY,        /// Current Battery level when using Raw Oxymeter input - value between 0 and 100///
    SA_HR_AMP,         /// heart amplitude information - 1st value between 0 and 100 -(end) get & getPoints & getVector ///
    SA_PULSE_VAR,      ///  Pulse rate information - couple of value between -400 and 400 and list of couple - getLorenz & getPoints ///
    SA_RR,             ///  Respiratory rate information - value between 0 and 45 - get & getPoints ///
    SA_RR_STATUS,      ///  Respiratory rate status information - couple of coordinate (SPO2 & RR) - getPoints ///
    SA_STRESS_CUR,     ///  Current stress information -  value between 0.7 and 2.5 and  0 and 1 - (end) get & getPoints (only one element in the list val, percentage) ///
    SA_STRESS_PARA,    ///  parasympathic stress information -  value between -5 and 5 and 0 and 1 - (end) get & getPoints (only one element in the list val, percentage)  ///
    SA_STRESS_ORTH,    ///  orthosympathic stress information -  value between -5 and 5 and  0 and 1 - (end) get & getPoints (only one element in the list val, percentage)  ///
    SA_IBI,            ///  Interbeat interval - couple (unfiltred, filtred IBI) - getPoints ///
    SA_HRV,            ///  Heart Rate Variability information - list of 50 values - getVector ///
    SA_AE,             ///  Arterial Elasticity - Type and then full information - get & getVector  ///
    SA_PPG_RAW,        ///  PPG signal from oximeter - getPoints only live  ///
    SA_PPG_FIL,        ///  PPG signal from oximeter filtred - getPoints only live  ///
    SA_ARRHYTHMIA,     ///  Arrhythmia evaluation - get (live & end ) & getPoints (end) ///
    SA_TIME,           ///  Current time - get (live)
};
// Getter - to call after a tick
// To call to get the current value of the data
bool isFinger() const; // if false need to stop recording as finger is not detected
std::pair<double,double> getLorenz() const; // -1000, -1000 if invalid value (value should be between -400 and 400

double get(DataType type) const; // -1 if invalid value or type
std::vector<std::pair<double,double>> getPoints(DataType type) const;
```

5.2.4.2. Signaux finaux

Une fois l'acquisition finie, des valeurs complémentaires sont accessibles et des valeurs moyennes et médianes sont calculés :

- get(DataType) permet de récupérer les valeurs. Si une valeur n'est pas disponible -1 sera renvoyé. Voici la liste des éléments disponibles :
    - SA_AE : type d'élasticité artériel (0=A, 5=F)
    - SA_ARRHYTHMIA : valeur finale de l'indicateur d'une potentielle arythmie

  - SA_HR : valeur médiane du rythme cardiaque
  - SA_HR_AMP : pourcentage de bonne amplitude
  - SA_PI : valeur moyenne de l'indicateur de perfusion (circulation générale du sang)
  - SA_RMSSD : valeur finale recalculé après filtrage de la variabilité cardiaque
  - SA_RR : valeur médiane du rythme respiratoire
  - SA_SPO2 : valeur médiane du niveau de saturation en oxygène
  - SA_SQI : valeur finale de l'indicateur de qualité du signal
  - SA_STRESS_CUR : niveau de stress actuel
  - SA_STRESS_PARA : niveau de stress parasympathique
  - SA_STRESS_ORTH : niveau de stress (ortho)sympathique
- getPoints(DataType) permet de récupérer une liste de points. Si une valeur n'est pas disponible un vecteur vide sera renvoyé. Voici la liste des éléments disponible en temps réel :
  - SA_ARRHYTHMIA : liste des valeurs d'arythmie identifiées au format (temps, valeur)
  - SA_HR : liste des valeurs du rythme cardiaque au format (temps, valeur)
  - SA_HR_AMP : 3 couples de valeur (bonne/moyenne/mauvaise amplitude, pourcentage)
  - SA_IBI : liste des intervalles entre les battements sous la forme de couples (valeur brute, valeur filtrée)
  - SA_PULSE_VAR : liste des couples de points pour la variabilité instantanée du rythme du pouls (ne fonctionne pas si vous faites la lecture rapide d'un fichier)
  - SA_SPO2 : liste des valeurs du taux de saturation en oxygène au format (temps, valeur)
  - SA_STRESS_CUR : couple (valeur, pourcentage de stress) pour le stress courant
  - SA_STRESS_PARA : couple (valeur, pourcentage de stress) pour le stress parasympathique
  - SA_STRESS_ORTH : couple (valeur, pourcentage de stress) pour le stress (ortho)sympathique
  - SA_RR : liste des valeurs du rythme respiratoire au format (temps, valeur)
  - SA_RR_STATUS : liste de couple de valeurs (SPO2, RR)
- getVector(DataType) permet de récupérer une liste de valeur
  - SA_AE : ensemble de paramètres en lien avec l'onde de pouls (la première valeur est le type identifié)
  - SA_HR_AMP : ensemble des amplitudes identifiées sur le signal
  - SA_HRV : ensemble de paramètres calculés pour la variabilité du rythme cardiaque
-

```
enum DataType{
    SA_PI=0,        /// Perfusion - Index value between 0 and 20 ///
    SA_SQI,         /// Signal Quality Index - value between 0 and 100 ///
```

```
    SA_HR,          /// Heart Rate - value between 0 and 220 - get & getPoints ///
    SA_SPO2,        /// Oxygen Saturation - value between 0 and 100 - get & getPoints ///
    SA_RMSSD,       /// Heart Variability of the total measurement - value between 0 and 400 ///
    SA_BATTERY,     /// Current Battery level when using Raw Oxymeter input - value between 0 and 100///
    SA_HR_AMP,      /// heart amplitude information - 1st value between 0 and 100 -(end) get & getPoints & getVector ///
    SA_PULSE_VAR,   /// Pulse rate information - couple of value between -400 and 400 and list of couple - getLorenz & getPoints ///
    SA_RR,          /// Respiratory rate information - value between 0 and 45 - get & getPoints ///
    SA_RR_STATUS,   /// Respiratory rate status information - couple of coordinate (SPO2 & RR) - getPoints ///
    SA_STRESS_CUR,  /// Current stress information - value between 0.7 and 2.5 and 0 and 1 - (end) get & getPoints (only one element in the list val, percentage) ///
    SA_STRESS_PARA, /// parasympathic stress information - value between -5 and 5 and 0 and 1 - (end) get & getPoints (only one element in the list val, percentage) ///
    SA_STRESS_ORTH, /// orthosympathic stress information - value between -5 and 5 and 0 and 1 - (end) get & getPoints (only one element in the list val, percentage) ///
    SA_IBI,         /// Interbeat interval - couple (unfiltred, filtred IBI) - getPoints ///
    SA_HRV,         /// Heart Rate Variability information - list of 50 values - getVector ///
    SA_AE,          /// Arterial Elasticity - Type and then full information - get & getVector ///
    SA_PPG_RAW,     /// PPG signal from oximeter - getPoints only live ///
    SA_PPG_FIL,     /// PPG signal from oximeter filtred - getPoints only live ///
    SA_ARRHYTHMIA,  /// Arrhythmia evaluation - get (live & end ) & getPoints (end) ///
    SA_TIME,        /// Current time - get (live)
};
// Getter - to call after a stop

double get(DataType type) const; // -1 if invalid value or type
std::vector<std::pair<double,double>> getPoints(DataType type) const;
std::vector<double> getVector(DataType type) const;
```

5.2.5. A propos

Pour avoir les informations relatives à la bibliothèque, la méthode « about » est disponible.

```
enum About{
  VERSION=0,
  COMPANY,
  UDI,
  RELEASE_DATE
};
// About
std::string about(const About &item);
```

5.3. Exemple 2 : avec la passerelle BridgeLibCardio

C'est une version plus bas niveau qui permet d'ajouter des callbacks pour récupérer des données à la volée et offre plus de flexibilité. Elle n'est pas recommandée par défaut.

5.3.1. Etape 1 : Initialisation du processus

Tous les encadrés de cette section son extrait du fichier à inclure *bridgeLibCardio.h*

```
enum DeviceType{
   BERRY,
   NONIN,
   FILE,
   RAW
};

///
/// \brief (Re)Initialize the LibCardio library.
///  Must be called once at the beginning of the program or when changing of device.
/// \return false if the initialisation failed without throwing an exception
///
bool init(DeviceType type, std::string source, unsigned int nbThreads=4);
/**
    * Start a processing and all working threads.
    * @param CaptureFs     Capture Frequency in Hz (should be 200Hz)
    * @param CoreFs       Core Frequency in Hz (should be 500Hz)
    */
   void start(int CaptureFs, int CoreFs);
```

La méthode à appeler pour initialiser la bibliothèque est **init**(…); qui intialise ou réinitialise les paramètres internes de la bibliothèque pour un type de dispositif et une source.

Pour ensuite démarrer les processus de la bibliothèque, vous devez appeler start(…);

5.3.2. Etape 2 : Envoyer les données à la bibliothèque

```
///
/// \brief To be called to retrieve the last packets from connected device
and process.
/// \param isFastReading : if the connected device is a file set to true
to read all the file in one go
/// \return false if the tick failed without throwing an exception
///
```

```
bool tick(bool isFastReading=false);

///
/// To be called to retrieve the last packets from raw mode.
/// \param rawData   All the captured data since the last tick in
hexadecimal.
/// \return false if the tick failed without throwing an exception
/// \note data are managed by group of 20
///
bool tick(const std::vector<uint8_t>& rawData);
///
/// To be called to retrieve the last packets from raw mode.
/// \param rawData   All the captured data since the last tick - could be
in hexadecimal or decimal.
/// \return false if the tick failed without throwing an exception
///
bool tick(const std::vector<std::string>& rawData, bool isDecimal =
false);
///
/// To be called to retrieve the last packets from raw mode.
/// \param rawData   All the captured data since the last tick in
hexadecimal.
/// \param size : Number of captured Data.
/// \return false if the tick failed without throwing an exception
///
bool tick(uint8_t* rawData, int size);
```

La méthode tick(…) envoie les données à la bibliothèque :

- tick(bool) sera utilisé avec tous les types de dispositif sauf les données brutes (si on lit un fichier, on pourra profité du mode lecture rapide qui ne donnera pas accès aux valeurs temps réelles mais permettra un calcul rapide des valeurs finales).
- tick(…) permettent d'envoyer des données brutes au format hexadécimal ou décimal à la bibliothèque (selon la méthode utilisée).

Une fois que les données sont reçues par la bibliothèque, les valeurs temps réelles sont disponibles.

5.3.3. Etape 3 : Arrêter la bibliothèque

Il est obligatoire d'arrêter la bibliothèque avant de quitter l'application car sinon, elle continuera à tourner en tâche de fond et consommera de la mémoire inutilement.

Pour arrêter le procesus, il faut appeler la méthode stop();

5.3.4. Etape 4 (optionnelle): Enregistrer des function de rappel (callback)

Pour acer des aux données d'affichage de graphe en temps reel, on peut utiliser la méthode LibCardio::registerDataCallback(…) qui permet de s'abonner à une notification. Dès qu'une valeur est disponible, la fonction enregistrée est appelée.

Cette méthode est à insérer entre l'appel des fonctions LibCardio::init() et LibCardio::start().

```
// static method which will be called by the callback
static void newRawPlotValues(LibCardio::LiveDataValue ldv)
{
   QVector<double> x(ldv.raw_plot.size);
   ::memcpy(x.data(), ldv.raw_plot.x, ldv.raw_plot.size*sizeof(double));
   QVector<double> y(ldv.raw_plot.size);
   ::memcpy(y.data(), ldv.raw_plot.y, ldv.raw_plot.size*sizeof(double));
  free(ldv.raw_plot.x);
   free(ldv.raw_plot.y);

    // do something now with the retrieved data
);

....
// Insert between the init and the start the registerDataCallback
void Backend::start(…)
{
   LibCardio::init(…);
   LibCardio::registerDataCallback(LibCardio::LIVE_RAW_PLOT,&newRawPlotValues);
   LibCardio::start(200, 500);
```

5.3.5. Etape 5 : Récupérer les valeurs d'un signal

5.3.5.1. Signaux temps réels

Pour récupérer les dernières valeurs disponible, il faut utiliser la méthode LibCardio::getLastData(…).

Voici un exemple de récupération d'une valeur temps réelle, l'indice de perfusion :

```
   LibCardio::LiveDataValue ldv;
   int pi=0;
   if(LibCardio::getLastData(LibCardio::LIVE_PI, &ldv))
   {
      pi = ldv.pi;
```

```
    }
```

Si la valeur que vous souhaitez récupérer et une valeur à tracer (plot), il faut faire attention aux fuites mémoires. La bibliothèque alloue un espace avec les données qu'il faudra libérer manuellement quand elles ne sont plus utilisées.

L'exemple suivant présente comment faire :

```
LibCardio::LiveDataValue ldv;
double ppg_fil = 0;
double ppg_fil_time = 0;

if(LibCardio::getLastData(LibCardio::LIVE_FIL_PLOT, &ldv)){
        ppg_fil = *ldv.fil_plot.y;
        ppg_fil_time = *ldv.fil_plot.x;

  free(ldv.fil_plot.y);
  free(ldv.fil_plot.x);

}
```

5.3.5.2. Signaux finaux

Une fois l'acquisition finie, 2 types de données supplémentaires sont disponibles :

- Les données globales : ce sont les données acquises tout au long de l'acquisition pour un paramètre (par exemple le rythme cardiaque)
- Les données qui ne sont accessibles qu'en fin de processus

Elles sont accessible en appelant la méthode LibCardio::**getFinalData**(…).

Cette méthode fonctionne exactement comme la méthode précédente avec toujours le besoin de libérer la mémoire pour les listes de points.

```
LibCardio::FinalDataValue fdv;
double *hr_x = 0; // time
double *hr_y = 0; // hr values

if(LibCardio::getLastData(LibCardio::FINAL_HR_PLOT, &fdv)){
        hr_x = fdv.hr.x;
        hr_y = fdv.hr.y;
```

```
    for(int i=0; i<val.ibi.size; i++){
      std::cout<<"HR["<<i<<"] ="<<hr_x[i]<<" , "<<
              hr_x[i]<<std::endl;
    }

    free(hr_x);
    free(hr_y);

     }
```

5.3.6. A propos

Pour avoir les informations relatives à la bibliothèque, la fonction « **about** » est disponible

```
enum About{
  VERSION=0,
  COMPANY,
  UDI,
  RELEASE_DATE
};
// About
std::string about(const About &item);
```

6. Intégration Java

Vous trouverez ici un exemple d'intégration en JAVA utilisant Eclipse. Il existe d'autres moyens d'intégrer la bibliothèque C++ dans un projet JAVA qui ne sont pas décrit ici.

6.1. Prérequis

- IDE Eclipse avec les plugins JDT et CDT
- JDK pour la compilation du JAVA
- Compilateur C++
    - Pour windows: Cygwin ou MinGW 64 pour compiler le code C++ Code (installer seulement l'un des compilateur pour éviter les problèmes de dépendances)

6.2. Étape 1 : Initialiser le projet Java

1. Démarrer Eclipse IDE
2. Nouveau projet Java
3. Sélectionnez le nom du projet :
    1. Il sera nommé « **HelloWorld_Project** » pour cet exemple.
    2. Sélectionnez **Ensuite**, puis **Terminez** pour créer le projet.
4. Dans le Package Explorer étendre le projet « **HelloWorld_Project** »
5. Cliquez à droite sur le dossier src et sélectionnez **New** > **Package**.

6. Entrez le nom comme **com.sensoriaanalytics.jni** puis cliquez sur **Fin**.
7. Dans le Package Explorer sous le « **HelloWorld_Project>src** » faire un clic droit
8. Entrez le nom comme « **HelloWorld** » et cliquez sur **Fin**.
9. La classe doit être définie comme la structure du code suivant.

```
1
2  //Package name
3  package com.sensoriaanalytics.jni;
4
5  //Class definition
6  class HelloWorld {
7
8      //In this part you define the native methods that are going to be imported from the C++ wrapper
9      public native void sayHello();
10
11     //We load the shared library here
12     static {
13         //You should only put the name of the library not the path to it
14         System.loadLibrary("HelloWorld");
15     }
16
17     public static void main(String[] args) {
18         HelloWorld h = new HelloWorld();
19         //Call the native methods
20         h.sayHello();
21     }
22
23 }
```

10. Allez au menu Eclipse IDE et sélectionnez « **Run > External Tools > External Configurations** ».
11. Sélectionnez le programme dans la liste dans le panneau de gauche.
12. Appuyez sur le nouveau bouton.
13. Entrez le nom comme « **javah – C Header and Stub File Generator**».
14. Trouvez l'emplacement de javah.exe: parcourir pour le localiser dans le dossier d'installation JDK (le fichier doit être sur un chemin similaire à c:\Program Files\Java\jdk1.7.0\bin\javah.exe)
15. Entrez le dossier de travail sous le nom de « **${project_loc}/bin/** »
16. Entrez les arguments sous le nom de « **-jni ${java_type_name}** » puis **appliquez**
17. Maintenant, à partir de l'onglet Commun, sélectionnez la case à cocher à côté des **outils externes** sous affichage dans le menu favoris, puis **appliquez et fermez**
18. **Désélectionner** Construire automatiquement à partir du menu du projet
19. Dans le **Package Explorer**, cliquez à droite sur « **HelloWorld_Project** » et sélectionnez « **Build Project** ».
20. Dans l'explorateur de paquets, mettez en surbrillance **HelloWorld.java** puis sélectionnez « **Run > External Tools > 1 javah – C Header and Stub File Generator**»
21. Cette étape générera le fichier en-tête pour le code C++ **com_ sensoriaanalytics _jni_HelloWorld.h** placé dans le dossier bin du projet Java « **HelloWorld_Project** »

6.3. Étape 2: la partie C++

1. À partir du menu sélectionnez **Fichier>Nouvel>Projet** puis développez C/C++

2. Mettre en surbrillance le projet C++ puis cliquez sur **Suivant**
3. Sous **le type de projet**, agrandir la **bibliothèque partagée** puis mettre en évidence le **projet vide**
4. Sous **Toolchains** sélectionnez **Cygwin GCC (ou MinGW GCC)**
5. Entrez le nom du projet comme **C_HelloWorld** appuyez **ensuite** sur **Suivant**
6. Décochez **Debug**. Ne laissez que la **Release** reste sélectionnée, puis cliquez sur **Fin**.
7. Si on vous demande de passer de Perspective à C/C++, **vérifiez et cliquez sur Oui**
8. Dans **le projet Explorer**, cliquez à **C_HelloWorld** et sélectionnez le fichier **New>Source**
9. Entrez le fichier **Source com_sensoriaanalytics_jni_HelloWorld.cpp** puis **terminez**.
10. Maintenant, écrivez l'emballage (wrapper) C++ suivant cette structure :

```
1  //Must include jni.h and the generated header from the java code
2  #include <jni.h>
3  #include "..\HelloWorld_Project\bin\com_sensoriaanalytics_jni_HelloWorld.h"
4  #include <stdio.h>
5
6
7  //Follow this format to create native methods
8  JNIEXPORT void JNICALL Java_com_sensoriaanalytics_jni_HelloWorld_sayHello(JNIEnv *env, jobject obj){
9      printf("Hello world!\n");
10     return;
11 }
```

11. Dans le projet Explorer mettre en **C_HelloWorld,** puis appuyez sur **Alt +Enter** pour ouvrir **les** propriétés pour le projet
12. Ouvrir **C/C++ Build**, highlight Settings, **selectionne Includes** de la liste dans l'onglet Paramètres d'outil sur le côté droit .
13. Cliquez **sur Ajouter**, **Système de** fichiers , puis parcourir le dossier inclure dans le module d'installation JDK et cliquez sur OK, (doit être quelque chose comme C:\Program Files\Java\jdk1.7.0\include).

14. Répète 13 mais cette fois inclure le dossier inclure \win32
15. Mettre **en évidence Divers sous Cygwin C Linker** (ou **MinGW C Linker**) de la liste
16. Sur le côté droit, entrez les drapeaux Linker **comme -mno-cygwin -Wl,--add-stdcall-alias** si vous utilisez Cygwin, autrement mis: **-Wl,--add-stdcall-alias**
17. Mettre en **surbrillance** les bibliothèques **sous le Linker C++**
18. Ajouter une bibliothèque (-l) : mettre le nom du fichier dll (sans l'extension .dll)
19. Ajouter un chemin de bibliothèque (-L) : placez le chemin d'annuaire où se trouvent les fichiers dll et .h.
20. Mettre en surbrillance **l'environnement sous C/C++ Build**
21. Editer la variable de chemin en ajoutant à la fin « ; (**chemin vers le fichier DLL et .h**) »
22. Ouvrir **C/C++ General**, puis **Paths** et **Symbols, GNU C++ in Includes** tab**.** Cliquer et Ajouter bouton. Ajouter le panneau de chemin d'annuaire apparaît, puis fournir la voie vers **la dll + fichiers en-tête.**
23. Passez à **l'onglet Artefact** Build sous **C/C++ Build**
24. Supprimer le texte pour le nom de l'artefact et taper **dans HelloWorld**

25. Supprimez le texte pour le préfixe de sortie et gardez-le vide
26. Cliquez à **droite C_HelloWorld** dans Project Explorer et sélectionnez **Build Project**

6.4. Étape 3 : lier l'emballage (wrapper) au code java

1. Passez à Java Perspective en sélectionnant dans le menu **Fenêtre>OpenPerspective>Other,** sélectionnez **Java (par défaut)** et cliquez sur OK
2. Dans le menu, sélectionnez Configurations Run>Run…
3. De la liste dans le volet gauche mettre en évidence **HelloWorld** sous **application Java**
4. Sur le côté droit, passez à **l'onglet** Arguments
5. Entrez **les arguments VM comme -Djava.library.path="${workspace_loc}\C_HelloWorld\Release »**
6. Dans le package explorer mettre en **évidence HelloWorld_Project**
7. Sélectionnez l'application **Run>Run As>Java**

6.5. Étape 4 : Un exemple de présentation

Le code fourni dans le fichier postal contient 2 dossiers, l'un pour la partie java et l'autre pour la partie de code natif.
Le code natif contient un fichier qui est l'emballage JNI et un dossier séparé qui contient la bibliothèque libCardio.

Il y a quelques méthodes qui devraient être appelés pour initialiser la bibliothèque et elles sont groupées cette méthode :

```
//A method that must be called first and only once to initiate the main functionalities of the lib
JNIEXPORT void JNICALL Java_com_lithiumhead_jni_HelloWorld_InitCardioS(JNIEnv *env, jobject obj, jint duration){

    int native_duration = (int)duration;

    printf("Initializing CardioS!\n");
    //Initialize the library
    CardioS::init();
    //Define the protocol
    CardioS::SerialProtocol a = CardioS::CARDIOS_PROTOCOL;
    //Start the raw data capture
    CardioS::startRaw(200, 500, native_duration, a);
    printf("Finished Initialization\n");

    return;
}
```

Et pour l'appeler à partir de Java, la méthode suivante est utilisée :

```
//Define the native methods
public native void InitCardioS(int duration);
```

Où la durée est la durée de l'enregistrement.

Dans la partie java, l'ensemble du fichier d'entrée est lu, puis il est envoyé segment par segment à l'emballage (parties de 10 lignes pour fournir à la bibliothèque suffisamment de données pour calculer les paramètres).

```
float[] results = h.callCardiosRaw(buffered_data);
```

Les données tamponnées sont en fait un tableau qui contient les octets lus à partir du fichier. Et le type d'adaptation se produit dans le côté de l'emballage :

```
//A method that takes the elements of the buffer(String array) then transforms them into uint8_t elements and pass it to cardioS lib then prints the live data
JNIEXPORT jfloatArray JNICALL Java_com_lithiumhead_jni_HelloWorld_callCardiosRaw(JNIEnv *env, jobject obj, jobjectArray stringArray){

    int length = env->GetArrayLength(stringArray);


    std::vector<uint8_t> bufferUInt8 = {};
    const char* cpp_string;

    //Read the elements of the input array one by one and transforms them into uint8_t
    for (int i = 0; i < length; ++i)
    {
        //Extract the data from the input jobjectArray and put it in a jstring
        jstring jstr = (jstring) env->GetObjectArrayElement(stringArray, i);
        //Convert the jstring to a cpp native string
        cpp_string = env->GetStringUTFChars(jstr, NULL);;

        //convert a const char* to an uint8_t* (Because it's the input type for the library)
        uint8_t* p = hex_str_to_uint8(cpp_string);
        bufferUInt8.push_back(*p);

        //Release the data held by the stringArray
        env->ReleaseStringUTFChars(jstr, cpp_string);
        env->DeleteLocalRef(jstr);
    }
    std::cout<<std::endl;

    //Put all the converted elements into an array
    uint8_t captureRawData[bufferUInt8.size()] ;
    std::copy(bufferUInt8.begin(), bufferUInt8.end(), captureRawData);
```

Les données dans buffered_data se transforme en type uint8_t puis enregistrées dans une liste uint8_t qui est l'entrée de libCardio.

```
//Put all the converted elements into an array
uint8_t captureRawData[bufferUInt8.size()] ;
std::copy(bufferUInt8.begin(), bufferUInt8.end(), captureRawData);
```

```
//Send the captured data to the library
CardioS::captureTickRaw(captureRawData, bufferUInt8.size());
```

Ensuite, pour capturer les données en direct, nous devrions appeler getlastdata puis obtenir la valeur que nous recherchons. Par exemple, voici comment récupérer les données ppg_filtered

données :

```
CardioS::LiveDataValue dat;

if(CardioS::getLastData(CardioS::LIVE_FIL_PLOT, &dat)){
    ppg_fil = *dat.fil_plot.y;
    ppg_fil_time = *dat.fil_plot.x;

    free(dat.fil_plot.y);
    free(dat.fil_plot.x);

}
```

The Spo2:

```
if(CardioS::getLastData(CardioS::LIVE_SPO2, &dat)){
    spo2 = dat.spo2;
}
```

NB : spo2, ppg_fil ppg_fil_time et les autres valeurs extraites sont tous des reels.

Puis enfin les données sont stockées dans un jfloatArray et renvoyer du côté java:

```
// Create an array that will hold the live data
jfloatArray result;
 result = env->NewFloatArray(9);
 if (result == NULL) {
    return NULL; /* out of memory error thrown */
 }

 //Copy the data to the output array
 std::vector<float> vec = {ppg_raw_time, ppg_raw, ppg_fil_time, ppg_fil, spo2, hr_time, hr, rr_time, rr};
 jfloat jvec[9];

 for(int i =0 ; i<9; i++){
    jvec[i] = vec[i];
 }
 env->SetFloatArrayRegion(result, 0, 9, jvec);

return result;
```

Ensuite, l'ensemble du processus est répété jusqu'à ce que toutes les données soient envoyées à la bibliothèque.

NB:

- **Plus les données sont récupérées de l'emballage les temps de calcul sont plus longs, c'est pourquoi une pause(10) a été utilisé entre tous les deux appels vers la bibliothèque du côté java sachant que JNI a quelques problèmes avec les appels successifs et rapides.**

7. Intégration sur application mobile

Nous présenterons ici l'étape d'intégration à l'aide de l'application basée sur ReactXP en utilisant Djinni-React-Native. Il existe désormais d'autres méthode d'intégration plus moderne mais que nous n'avons pas encore testé n'étant pas spécialisé dans le mobile – cependant, nous pouvons vous aider dans l'intégration qui utilisent d'autres méthodes si vous avez besoin.

Si une autre application de développement mobile est utilisée, il faudra adapter certaines de ces étapes.

7.1. ReactXP - Djinni

Les prérequis pour créer et exécuter une application sont ceux de ReactXP.

Les étapes suivantes fonctionneront avec Android et iOS.

Nous utilisons une couche CPP multiplateforme de bas niveau.

La communication avec la partie Javascript de ReactXP se fait via un code « Bridge » qui a été généré avec Djinni-React-Native. Le fichier idl et le script pour générer la définition du « Bridge » se trouvent dans le dossier generated_src.

Si vous souhaitez connecter du Java et du CPP, vous pouvez utiliser Djini avec le fichier IDL ou d'autres outils à votre convenance et expertise.

Veuillez noter que l'intégration de la bibliothèque est hybride en ce sens qu'avant l'existence du protocole LIBCARDIO, nous avions besoin d'une approche de base (Démarrer / Envoyer les données de l'oxymètre / Arrêter / Recevoir les données finales). C'est pourquoi l'application n'utilise qu'un petit sous-ensemble de la surface API disponible. Toutes les données devant être affichées à l'écran étaient disponibles via les paquets entrants.

Avec le protocole LIBCARDIO, les données PlethWave ne sont plus disponibles et doivent être demandées à la bibliothèque CPP LibCardio. Heureusement, nous utilisions déjà une approche flexible pour récupérer une carte Javascript des objets pour la fréquence du pouls, il est donc facile d'ajouter des données PlethWave à cette carte.

If you want to bridge Java and CPP, you could use Djini with the IDL file, or other tools.

Please note that the library integration is hybrid in the sense that before the existence of the LIBCARDIO protocol, we needed a basic approach (Start/Send Oximeter Data/Stop/Receive Final data). This is why the app only use a tiny subset of the API surface available. All the data required to be displayed on screen was available in the incoming packets.

7.1.1. Capture des données

Une fois que vous êtes connecté au périphérique que ce soit en USB, en BLE ou par un fichier rejoué :
Le procole utilisé pour decoder les données et determine en examinant l'entête des paquets.

```
// Berry protocol 1.3
// tslint:disable-next-line:no-bitwise
if (value[0] & oxff && value[1] & oxaa) {
} else {
        // BCI protocol 1.2
        // No checksum
}
```

Le protocole propriétaire intégré dans LIBCARDIO est détecté par une requête à l'oxymètre. Au contraire des protocoles BCI/BRI, la donnée PlethWave n'est alors plus accessible comme donnée d'entrée Il faut envoyer les paquets à la bibliohtèque et en retour, la donnée PlethWave sera accessible (comme présenté plus bas).

7.1.2. Le moteur de donnée en CPP

Le moteur de données a besoin d'au moins 50 paquets de 20 octets de données pour ses algorithmes. Une fois que nous avons accumulé suffisamment de données, elles sont envoyées pour traitement. Le moteur revient immédiatement avec la fréquence respiratoire actuelle, s'il a accumulé suffisamment de données pour la calculer. Si le protocole LIBCARDIO est utilisé, nous récupérons également les données PlethWave:

Ce qui suit est le code du « Bridge » CPP / Javascript :

```
// Send the data comming from the JS side to the engine

LibCardio::captureTick(data, nb);

// Prepare for returning data to the consuming application

auto map = mBridge->createMap();

// Query for Respiration rate

LibCardio::LiveDataValue liveRR;

if (LibCardio::getLastData(LibCardio::LIVE_RR, &liveRR)) {

} else {

        liveRR.rr = 0;
```

```
}
map->putInt("rr", liveRR.rr);
// Query for Pleth Wave plot data
LibCardio::LiveDataValue liveFillPlot;
if (LibCardio::getLastData(LibCardio::LIVE_FIL_PLOT, &liveFillPlot)) {
        // sounds good
        LOG_VERBOSE("< DataBridgeImpl::provideOxymeterData returning RR "
        "[%d] FillPlot size [%d] "
        "FillPlot X [%f] FillPlot Y [%f]",
        liveRR.rr, liveFillPlot.fil_plot.size,
        *liveFillPlot.fil_plot.x, *liveFillPlot.fil_plot.y);
} else {
        liveFillPlot.fil_plot.size = 0;
        liveFillPlot.fil_plot.x = nullptr;
        liveFillPlot.fil_plot.y = nullptr;
}
// Set the size of the arrays returned
if (liveFillPlot.fil_plot.size != 0) {
        map->putInt("fillPlotSize", liveFillPlot.fil_plot.size);
}
// populate the arrays
auto arrayliveFillPlotX = mBridge->createArray();
auto arrayliveFillPlotY = mBridge->createArray();
```

```
if (liveFillPlot.fil_plot.x != nullptr && liveFillPlot.fil_plot.y != nullptr) {
        for (int i = 0; i < liveFillPlot.fil_plot.size; ++i) {
            arrayliveFillPlotX->pushDouble(liveFillPlot.fil_plot.x[i]);
            arrayliveFillPlotY->pushDouble(liveFillPlot.fil_plot.y[i]);
        }
}
else
{
        LOG_VERBOSE("No Data for PlethWave");
}
map->putArray("fillPlotX", arrayliveFillPlotX);
map->putArray("fillPlotY", arrayliveFillPlotY);
// Resolve the promise to return the data to the consumin app.
promise->resolveMap(map);
```

Cependant pour la partie Javascript, nous devons traiter chaque paquet reçu comme présenté dans la section suivante.

7.1.3. Affichage de données pendant une capture

Tout ce qui est reçu est décodé et traité en temps réel. Mais le taux de rafraîchissement des données à l'écran est fixé à 1 seconde. Nous n'utilisons pas de minuterie pour savoir où muter l'état de l'écran. Nous utilisons la fréquence de capture pour savoir quand 1 seconde a été atteinte.

Cela signifie que les données peuvent être accumulées dans des tampons si nécessaire (principalement pour l'affichage graphique de PlethWave).

7.1.3.1. Le graphe PlethWave

Les données affichées sont gérées par une liste chaînée dont la longueur est égale à la fréquence de capture * 5. À 200 Hz, cela signifie que nous avons un maximum de 1000 points. Pour imiter le fait que le graphique se déplace vers la gauche lorsque la liste est pleine, nous supprimons la valeur la plus ancienne de la liste et ajoutons la nouvelle.

N'oubliez pas que lorsque vous utilisez le protocole LibCardio, les données proviennent du moteur CPP et non pas de l'appareil.

7.1.3.2. Le graphe du rythme cardiaque (Pulse Rate)

Les données affichées sont gérées par une liste chaînée dont la longueur est égale à 60 points (1 minute de données). Pour imiter le fait que le graphique se déplace vers la gauche lorsque la liste est pleine, nous supprimons la valeur la plus ancienne de la liste et ajoutons la nouvelle.

7.1.3.3. Le graphe de Lorenz

Les données affichées sont soutenues par une liste chaînée dont la longueur est égale à 20. Un maximum de 20 bulles sont affichées).

7.1.3.4. L'animation de respiration

Cette partie peut être basé sur du CSS et la durée est recommandé est de 10s.

7.1.3.5. Toutes les jauges

La valeur qui serait affichée est mise à jour en temps réel. Lorsque l'état de l'écran est muté, la valeur actuelle est utilisée.

7.1.3.6. Saving patient informations at the end of the capture

À la fin de la capture, il faut :

• Envoyer les données de configuration au moteur CPP (subject_age / subject_height)
• Arrêtez le moteur CPP. Il déclenche un événement pour renvoyer les données radar à l'application.
• Les données sont enregistrées sur le téléphone.
• Les données sont ensuite envoyées au compartiment cardiosensys-mobile AWS S3
• Les données accumulées sont ensuite enregistrées dans un fichier CSV

• Le fichier CSV est ensuite envoyé au compartiment cardiosensys-mobile AWS S3.
• Accédez à la vue radar

7.1.3.7. Affichage du radar final

Les données radar sont un fragment JSON qui contient tout ce dont nous avons besoin pour afficher le radar et chaque graphique individuel associé.

Il est renvoyé automatiquement lorsque vous arrêtez le moteur dans le code de pont en déclenchant l'événement NEW_RADAR_DATA.

```
// Stop the engine (this should be done first, otherwise the following code
// would crash the app)
LibCardio::stop();
// Get radar data
LibCardio::FinalDataValue val;
// set subject age and height
setRadarAge(mSubjectAge);
setRadarHeight(mSubjectHeight);
bool retval = LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_HR, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_HR, &val) "
"returned [%s]",
retval ? "true" : "false");
setRadarHR(val.hr.x, val.hr.y, val.hr.size);
retval = LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_RR, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_RR, &val) "
"returned [%s]",
retval ? "true" : "false");
setRadarRR(val.rr.x, val.rr.y, val.rr.size);
retval = LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_SPO2, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_SPO2, "
"&val) returned [%s]",
retval ? "true" : "false");
setRadarO2(val.rro2.x, val.rro2.y, val.rro2.size);
retval = LibCardio::getFinalData(LibCardio::FINAL_RRO2, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_SPO2, "
"&val) returned [%s]",
retval ? "true" : "false");
setRadarRRO2(val.rro2.x, val.rro2.y, val.rro2.size);
retval = LibCardio::getFinalData(LibCardio::FINAL_RMSSD, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_RMSSD, "
"&val) returned [%s]",
retval ? "true" : "false");
setRMSSD(val.rmssd);
```

```
retval = LibCardio::getFinalData(LibCardio::FINAL_FEATURES, &val);
LOG_VERBOSE("LibCardio::getFinalData(LibCardio::FinalDataType::FINAL_FEATURES, "
"&val) returned [%s]",
retval ? "true" : "false");
setRadarEAType(val.features.angle, val.features.cVal, val.features.bVal,
val.features.eaType);
setRadarIRA(val.features.ppgai, val.features.tsDic);
setRadarTA(val.features.pai);
// conversion: RadarData -> json
json j = radarData_;
std::string stringified = j.dump();
LOG_VERBOSE("RadarData JSON string is [%s]", stringified.c_str());
// Emit an event to the UI
auto ret = mBridge->createMap();
ret->putString("radarData", stringified);
mBridge->emitEventWithMap(DataBridge::NEW_RADAR_DATA, ret);
promise->resolveInt(0);
```

8. Problème de connexion de l'oxymètre Berry

8.1. Linux (Ubuntu)

La plupart du temps le problème provient des droits d'accès au pilote.

(vous devez avoir les droits administrateurs pour réaliser ces manipulations présentés dans la suite de ce document)

La meilleure façon de résoudre ce problème est de déterminer le groupe auquel appartient le dispositif et d'ajouter votre compte à ce dernier.

On suppose que le dispositif se connecte sur ttyACMo

```
# exploration du problème
ls -l /dev/ttyACMo
man ls
# resolution du problème
sudo adduser $USER $(stat --format="%G" /dev/ttyACMo )
```

Cette solution résout le problème pour un compte. Si vous souhaitez que d'autres comptes se connecte au dispositif, il vous faudra répéter l'opération pour chacun d'entre eux.

Une alternative sera d'ajouter une règle dans le repertoire */etc/udev/rules.d*.

Vous devrez tout d'abord lancer la fonction *dmesg* pour obtenir les informations sur le vendeur et l'identifiant du produit.

Puis vous pouvez créer un fichier nommé *99-sensoria.rules* (s'il y a déjà une règle avec le numéro 99, vous pouvez le changer par un nombre nom utilisé, vous pouvez aussi choisir un nom différent de sensoria si c'est plus parlant pour votre projet).

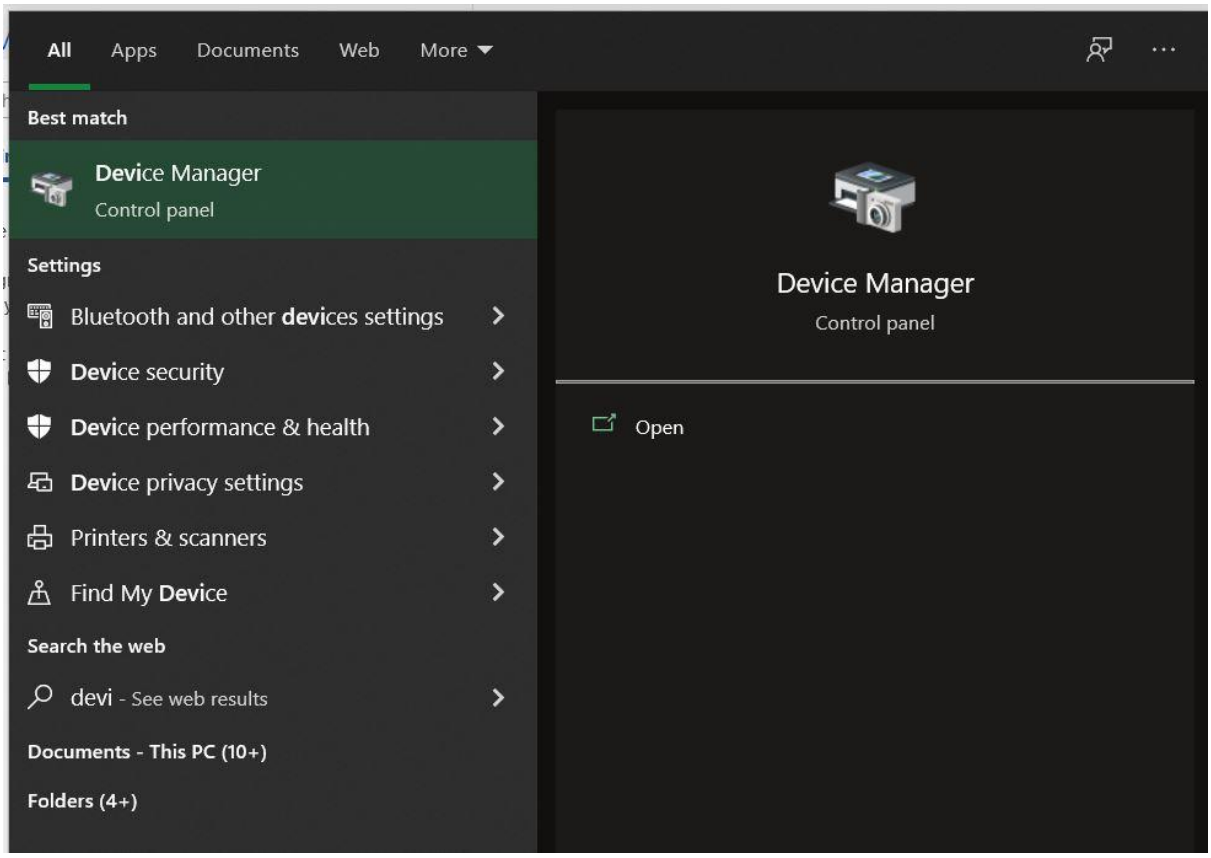
Contenu du fichier *99-sensoria.rules*

```
# USB-Sensoria
SUBSYSTEM=="usb", ATTR{idVendor}=="XXXX", ATTR{idProduct}=="*", \
ENV{DEVTYPE}=="usb_device",MODE="0666"
```

XXXX sera remplacé par l'identifiant du vendeur

8.2. Windows

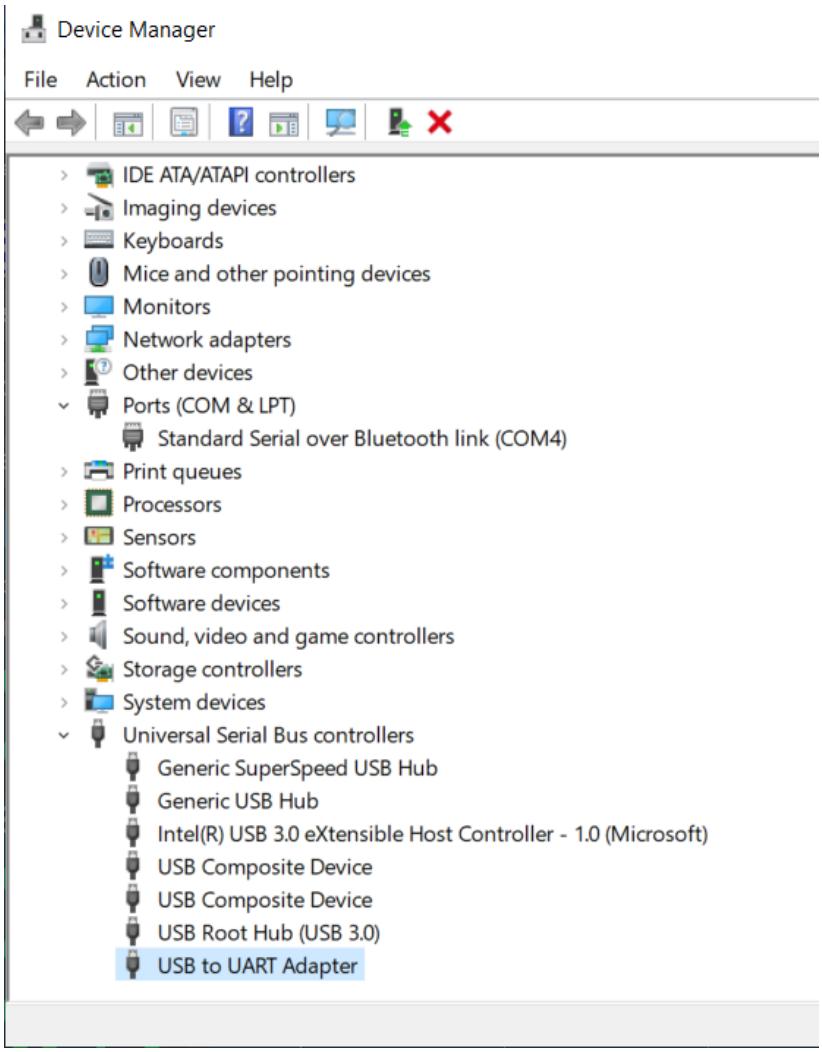
8.2.1. Ouvrir le gestionnaire de peripherique/device manager



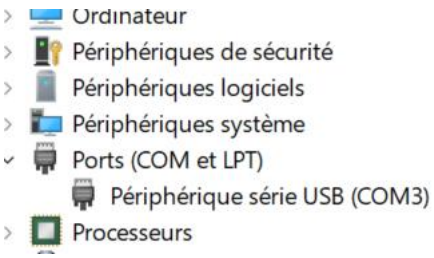
8.2.2. Rechercher le dispositif associé à l'oxymètre

- S'il apparait dans *Contrôleurs de bus USB/Universal Serial Bus controllers* (cherchez USB to UART Adapter), il faut mettre à jour le driver

Clic droit sur le USB to UART Adapter -> Propriété

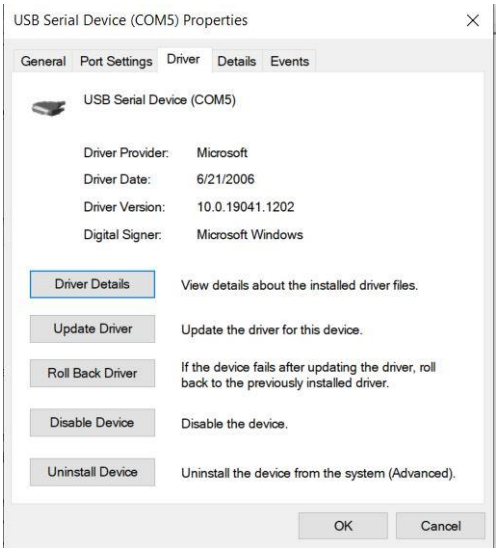


- Sinon il devrait apparaitre dans Ports (COM & LPT) sous le nom USB Serial Device (COM X), avec X un numéro de COM. Si vous avez plusieurs dispositifs dans la liste déconnectée puis reconnectée l'oxymètre pour identifier le bon port.

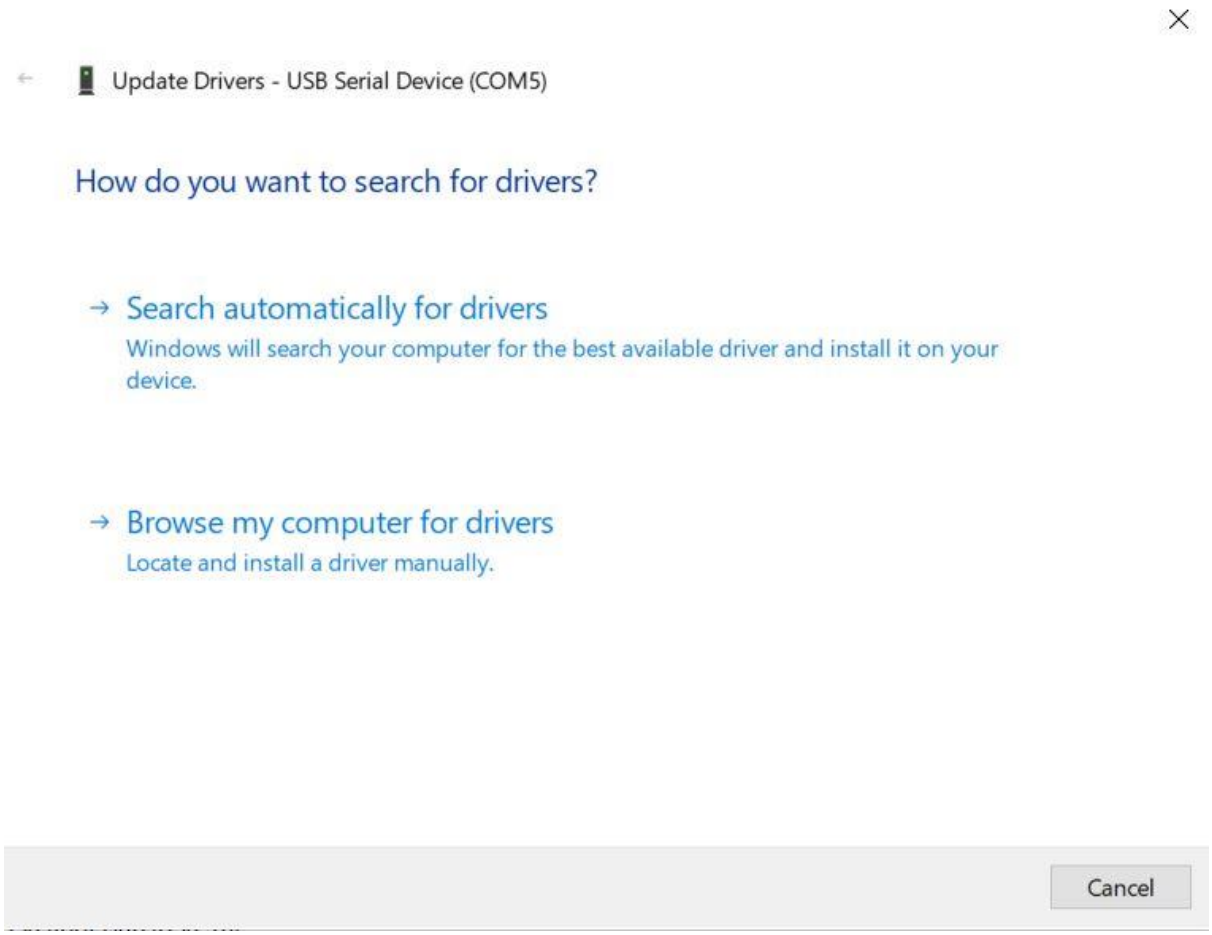


Clic droit sur le USB to COM X -> Propriété

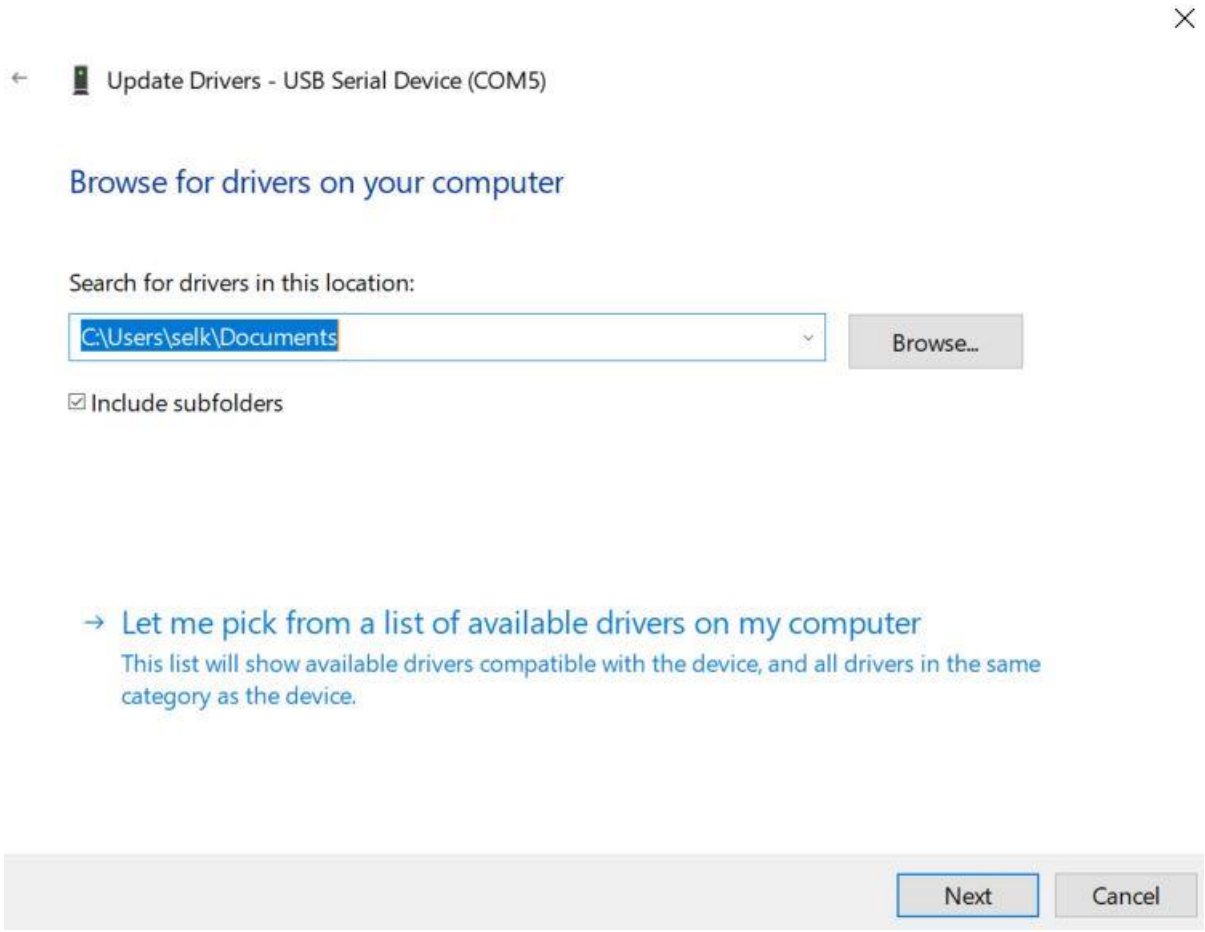
- Aller dans la section *Pilote/Driver* et cliquer sur *Mettre à jour le pilote/Update Driver*



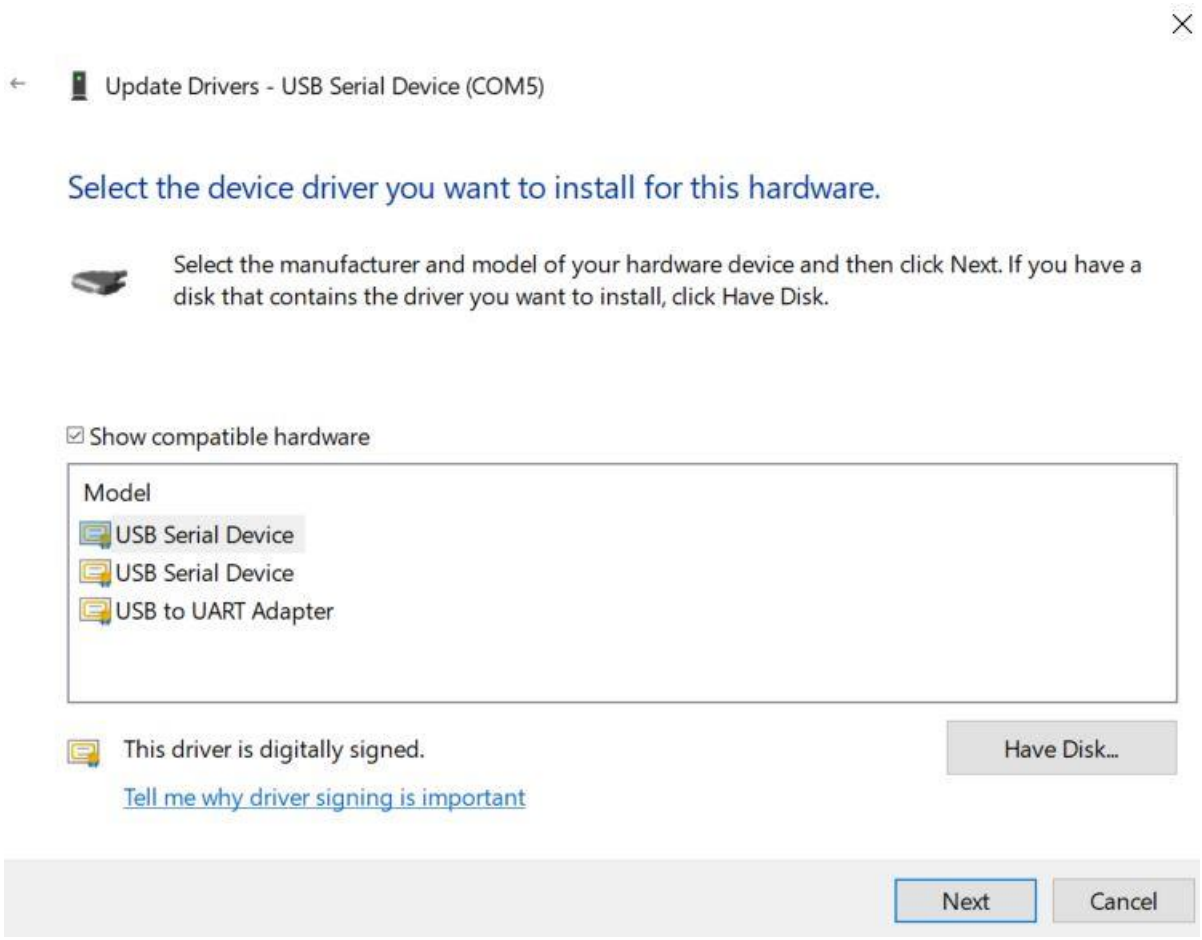
- Sélectionnez la seconde option (parcourir l'ordinateur pour rechercher les drivers)



- Sélectionner : *Laisser moi choisir à partir de la liste des drivers disponibles sur mon ordinateur / Let me pick from a list of available drivers on my computer.*



- Sélectionner le premier dispositif de la liste et cliquez sur suivant


Update Drivers - USB Serial Device (COM5)
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Show compatible hardware
Model
USB Serial Device
USB Serial Device
USB to UART Adapter
This driver is digitally signed.
Have Disk...
Tell me why driver signing is important
Next
Cancel

9. Label

FIN DU DOCUMENT