

Identité, horloge et marqueurs repas

La version d'origine savait récupérer les glycémies, et c'était déjà beaucoup. Mais pour le suivi de mon proche, il me fallait trois choses en plus : savoir **quel lecteur** on lit, savoir si son **horloge est juste**, et récupérer les **marqueurs repas**. Sans ces marqueurs, impossible de savoir quelle mesure est "la glycémie du matin à jeun", celle sur laquelle repose tout le protocole du médecin.

Ce chapitre, c'est un peu comme Tidepool ne m'a pas mâchée. Il a fallu repartir de la norme et des



vrais octets.

Décortiquer la carte d'identité du lecteur

Voici la réponse d'un vrai Accu-Chek Guide (modèle 925, firmware v1.9.6) à la demande d'attributs MDS. Le numéro de série, l'identifiant système et l'horloge ont été remplacés, mais la structure est d'origine :

E700 00D4 00D2 00A4 0203 00CC 0000 0008 00C6	en-tête : réponse GET, 8 attributs
0928 000E 0006 526F63686500 0004 39323500	ID_MODEL
0984 000A 0008 0060190000000042	SYS_ID
0A44 0002 4000	DEV_CONFIG_ID
092D 0056 0003 0052	ID_PROD_SPECN, 3 entrées :
0001 0000 001A 73657269616C2D6E756D6265723A20393235303030303030343200	
0003 1000 0012 68772D7265766973696F6E3A2047000000000000	
0005 0000 0014 66772D7265766973696F6E3A2076312E392E3600	
0987 0008 2026100121065800	TIME_ABS
0A45 0010 C000 1F00 FFFFFFFF 0064 000000000000	MDS_TIME_INFO
0A4B 0016 ...	REG_CERT_DATA_LIST
0A5A 0008 0001 0004 1011 0002	SYS_TYPE_SPEC_LIST

Chaque attribut suit le même schéma : 2 octets d'identifiant, 2 octets de longueur, puis la valeur.
Ligne par ligne :

- 02 03 : c'est bien la réponse à notre GET. 00 08 : elle contient 8 attributs.
- 09 28 (2344, ID_MODEL) : deux chaînes de caractères. 526F636865 en ASCII donne **"Roche"**, 393235 donne **"925"**.
- 09 84 (2436, SYS_ID) : l'identifiant système du lecteur sur 8 octets (un EUI-64), écrit en hexadécimal dans la sortie.
- 09 2D (2349, ID_PROD_SPECN) : une liste de "spécifications de production". Chaque entrée a un type : **1** pour le numéro de série, **3** pour la version matérielle, **5** pour le firmware. En ASCII : serial-number: 92500000042, hw-revision: G, fw-revision: v1.9.6. Roche met une étiquette devant la valeur, la fonction afterLabel garde ce qui suit le ":".
- 09 87 (2439, TIME_ABS) : l'horloge du lecteur, en BCD : **2026/10/01 21:06:58**.
- 0A 45 (2629, MDS_TIME_INFO) : les capacités de l'horloge. Les deux premiers octets, C0 00, sont des bits. 0x8000 veut dire "j'ai une horloge temps réel" et 0x4000 veut dire **"on peut me mettre à l'heure"**. C'est ce bit que teste le code.
- 0A 5A (2650, SYS_TYPE_SPEC_LIST) : le profil d'appareil. 10 11 (4113) correspond au profil "glucomètre" de la norme (ISO/IEEE 11073-10417), en version 2. La confirmation que l'on parle bien à un appareil standard, et pas à un protocole maison.

Le décodage est fait par parseMdsAnswer, qui parcourt les attributs un par un et ignore ceux qu'il ne connaît pas. Si un lecteur ne donne pas certains attributs, les champs restent vides, et si la réponse n'est pas compréhensible du tout, la sortie indique simplement "meter": null. On ne bloque pas le téléchargement des mesures pour une carte d'identité mal lue.

L'horloge : un détail qui n'en est pas un

Le lecteur date chaque mesure avec **sa propre horloge**. Si elle a 25 minutes d'avance, toutes les mesures ont 25 minutes d'avance. Et si elle n'a pas été changée au passage à l'heure d'hiver, une glycémie faite à 6h45 apparaît à 7h45... ou une mesure de 11h30 bascule après midi et sort de la plage "matin". Pour un protocole qui repose sur "la glycémie du matin", c'est gênant.

accucheek lit donc l'horloge du lecteur au début de la session et la compare à celle du PC. La sortie JSON le montre :

```
"clock": {"meter": "2026/10/01 21:06:58", "pc": "2026/10/01 20:42:52", "offset_s": 1446, "settable": true, "pc_synchronized": true, "action": "set"}
```

Ici, le lecteur avait 1446 secondes d'avance, soit un peu plus de 24 minutes.



Remettre le lecteur à l'heure, mais prudemment

Avec l'option `--set-time`, `accucheek` peut corriger l'horloge du lecteur. Le message est construit par `buildSetTime` : une action `0C 17` sur l'objet MDS (handle 0), avec la nouvelle date en BCD.

Écrire dans un appareil médical, ce n'est pas anodin. J'ai donc mis plusieurs garde-fous, testés dans cet ordre par `decideClock` :

1. on n'a rien demandé : on ne touche à rien (`not_requested`) ;
2. le lecteur n'a pas donné son heure : on ne sait pas, on ne touche à rien (`unknown`) ;
3. le lecteur ne déclare pas le bit "on peut me mettre à l'heure" : on ne touche à rien (`not_settable`) ;
4. l'heure du PC est inconnue : on ne touche à rien (`pc_unknown`) ;
5. l'heure du PC **n'est pas synchronisée par NTP** : on ne touche à rien (`pc_not_synchronized`). Pas question de recopier sur le lecteur l'heure d'un PC lui-même déréglé ;
6. l'écart est de 60 secondes ou moins : inutile, on laisse (`within_tolerance`) ;
7. sinon, on règle l'horloge (`set`).

Pour savoir si le PC est synchronisé, pas besoin d'être root : la fonction `systemClock` appelle `adjtimex` en lecture seule et vérifie que le noyau ne répond pas `TIME_ERROR`, c'est-à-dire que l'horloge est bien tenue à jour par NTP.

```
PcClock systemClock() {
    struct timex tx;
    memset(&tx, 0, sizeof(tx));    // modes = 0 : lecture seule, aucun privilège
    auto state = adjtimex(&tx);
    return {time(0), true, 0<=state && TIME_ERROR!=state};
}
```

Si le lecteur refuse quand même la mise à l'heure, le téléchargement continue normalement et la sortie indique `"action": "rejected"`. Une horloge pas corrigée, ce n'est pas une raison de perdre les mesures.

L'option `--now`, qui permet de simuler une heure de PC, n'est acceptée qu'avec `--replay` (le rejeu d'une trace enregistrée). Une fausse heure ne doit jamais pouvoir atteindre un vrai lecteur.

Les marqueurs repas

Sur l'Accu-Chek Guide, après une mesure, on peut indiquer le moment : à jeun, avant repas, après repas... Ces marqueurs ne sont pas rangés avec les glycémies, mais dans **leur propre segment** (`PMSegCMeal`). Chaque entrée fait 10 octets : la date en BCD et un code de la nomenclature ISO/IEEE 11073-10417 (le profil "glucomètre" de la norme).

Code	Hexa	Dans la sortie	Signification
29260	72 4C	before_meal	avant repas
29264	72 50	after_meal	après repas
29268	72 54	fasting	à jeun
29272	72 58	casual	moment quelconque
29300	72 74	bedtime	coucher
autre		other	code inconnu

Recoler chaque marqueur à sa mesure

Le problème, c'est qu'un marqueur ne dit pas "je vais avec la mesure n°42". Il a juste une date. La seule façon de le relier à sa glycémie, c'est l'heure, et pas à la minute près : **à la seconde près**, octets bruts compris.

Le code garde donc, pour chaque glycémie et chaque marqueur, les 8 octets de date tels quels dans un entier de 64 bits (`timeKey`). Ensuite, `attachMeals` range les glycémies par date dans une table de hachage et y cherche chaque marqueur :

```
std::unordered_map<uint64_t, std::vector<size_t>> byTime;
for(size_t i=0; i<samples.size(); ++i) {
    byTime[samples[i].timeKey].push_back(i);
}
for(const auto &m : meals) {
    auto it = byTime.find(m.timeKey);
    if(byTime.end()==it) {
        ++unmatched;    // marqueur sans mesure à la même seconde
        continue;
    }
```

```
for(auto i : it->second) {  
    samples[i].meal = m.meal;  
}  
}
```

Les marqueurs qui ne trouvent pas leur mesure ne sont pas inventés ni rattachés "à peu près" : ils sont comptés, et le total apparaît dans la sortie (`"unmatched": 1`). Dans la trace de test du chapitre précédent, il y a justement un marqueur "avant repas" daté du 29/09 à 08:00:00 sans aucune glycémie à cette seconde-là. Il ressort comme orphelin.

Une dernière précaution dans `parseMealSegment` : un paquet de glycémies lu par erreur comme un paquet de marqueurs pourrait "tenir" dans la mémoire. Le code vérifie donc que le nombre d'octets annoncé correspond exactement à "nombre d'entrées x 10". Si ça ne colle pas, on s'arrête avec un message clair.

HI et LO : les mesures hors échelle

Quand la glycémie est trop haute ou trop basse pour être mesurée, le lecteur affiche "HI" ou "LO". En mémoire, il ne stocke pas un nombre mais une valeur spéciale :

- `0x07FE` pour **HI** (au-dessus de la plage du lecteur) ;
- `0x0802` pour **LO** (en dessous).

La version d'origine les jetait. Or un "LO", c'est une hypoglycémie sévère : c'est sans doute la mesure qu'on a le moins envie de perdre ! Maintenant, elles sont gardées avec la convention de Tidepool : 601 mg/dL avec `"range": "high"`, ou 9 mg/dL avec `"range": "low"`. Le programme qui lit le JSON sait ainsi que ce n'est pas une vraie valeur mesurée, mais il sait qu'elle existe.

Dans le même esprit, **toutes** les mesures sont écrites, avec leur statut brut (0 pour une mesure normale). Une mesure dont la date n'est pas du BCD valide sort avec `"error": "invalid date"` au lieu de disparaître. C'est au programme qui lit la sortie de décider ce qu'il garde, pas à accuchek de trier en silence.

Le bug de l'heure d'été

Dernier point, et c'est un classique. Le lecteur donne une heure "locale", sans fuseau horaire. Pour la convertir en `epoch` (nombre de secondes depuis 1970), on utilise `mktime`, en lui passant une structure `tm`. Dans cette structure, il y a un champ `tm_isdst` qui dit si on est en heure d'été.

Le code d'origine le mettait à 0, ce qui veut dire "heure d'hiver", toute l'année. Résultat : toutes les mesures prises en été avaient un `epoch` décalé d'une heure. La correction tient en une ligne :

```
t.tm_isdst = -1;    // laisser les règles du fuseau horaire décider
```

Avec , c'est la base de fuseaux horaires du système qui décide, date par date, si on était en heure d'été ou non. Des tests vérifient ça noir sur blanc : une mesure affichée 07:36 le 2 juillet 2026 doit donner 05:36 UTC (heure de Paris en été), et des mesures juste avant et juste après les changements d'heure de mars et d'octobre doivent tomber au bon endroit. Le cas tordu de 02:30 le 25 octobre, une heure qui existe deux fois cette nuit-là, est aussi couvert : les deux réponses sont acceptées.

Révision #7

Créé 2026-10-02 20:28:08 UTC par Renard

Mis à jour 2026-10-03 12:44:12 UTC par Renard