Activité : Géolocalisation par GPS

Thierry Vaira <tvaira@free.fr>

01/02/2016 (rev. 1)

Table des matières

Géolocalisation par GPS

Notions de base

Le GPS (*Global Positioning System*) est un système de géolocalisation fonctionnant au niveau mondial et reposant sur l'exploitation de signaux radio émis par des satellites dédiés. Le GPS utilise le système géodésique WGS 84, auquel se réfèrent les coordonnées calculées grâce au système.

Lire : Global Positioning System et Récepteur GPS

Une personne munie d'un récepteur GPS peut ainsi se localiser et s'orienter sur terre, sur mer, dans l'air ou dans l'espace au voisinage proche de la Terre.

La norme NMEA 0183

La norme **NMEA 0183** est une spécification pour la communication entre équipements marins, dont les équipements GPS. Elle est définie et contrôlée par la *National Marine Electronics Association* (NMEA), association américaine de fabricants d'appareils électroniques maritimes.

La norme 0183 utilise une simple communication série pour transmettre une "phrase" (*sentence*) à un ou plusieurs écoutants. Une trame NMEA utilise tous les caractères ASCII.

Exemple : *Waypoint Arrival Alarm*

```
$GPAAM,A,A,0.10,N,WPTNME*32
```

Il existe plus d'une trentaine de trames GPS différentes. Le type d'équipement est défini par les deux caractères qui suivent le `$`. Le type de trame est défini par les caractères suivants jusqu'à la virgule.

Par exemple :

```
$GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E
```

C'est une trame ___GP___S de type **GGA**. La trame GGA est très courante car elle fait partie de celles qui sont utilisées pour connaître la position courante (longitude, latitude et altitude) du récepteur GPS (www.coordonnees-gps.fr).

- `$` : délimiteur de début de trame
- `GP` : Id du "parleur" (GPS)
- `GGA` : Type de trame (*Global Positioning System Fixed Data*)
- `064036.289` : Trame envoyée à 06h40m36,289s (heure UTC)
- `4836.5375,N` : Latitude ddmm.mmmm -> 48,608958° Nord = 48°36'32.25" Nord
- `00740.9373,E` : Longitude dddmm.mmmm -> 7,682288° Est = 7°40'56.238" Est
- `1` : Type de positionnement (le 1 est un positionnement GPS)
- `04` : Nombre de satellites utilisés pour calculer les coordonnées
- `3.2` : Précision horizontale ou HDOP (Horizontal dilution of precision)
- `200.2,M` : Altitude 200,2, en mètres
- `,,,,,0000` : D'autres informations peuvent être inscrites dans ces champs
- `*0E` : Somme de contrôle de parité, un simple XOR sur les caractères précédents
- `<CR><LF>` : délimiteur de fin de trame

Remarque : Les trames NMEA font toutes référence à l'ellipsoïde **WGS84** comme base de son système de coordonnées. Ce document détaille la conversion de coordonnées GPS.

Chaque trame a sa syntaxe propre, mais selon le cas elles peuvent ou doivent se terminer, après le `*`, par un octet formant une somme de contrôle (*checksum*) qui permet de détecter une erreur dans la transmission.

La somme de contrôle à la fin de chaque phrase est le **OU EXCLUSIF** (`XOR`) de tous les octets de la phrase à l'exclusion du premier caractère (`$`) et jusqu'au caractère avant l'étoile (`*`). Cf. C implementation of checksum generation.

- Site officielle : www.nmea.org
- Documentation : NMEA 0183

Réception de phrases NMEA 0183

Actuellement lorsque l'on raccorde un récépteur GPS USB à un ordinateur, cela revient à gérer un port série virtuel.

On peut tester la réception de phrases NMEA 0183 avec la commande `screen` (`Ctrl-a k` pour sortir) ou avec `picocom` (`Ctrl-a Ctrl-x` pour sortir) :

```
$ screen /dev/ttyACM0 115200
$GPRMC,000936.799,V,0000.0000,N,00000.0000,E,,,060180,,*10
$GPVTG,,T,,M,,N,,K*4E
$GPGGA,000937.799,0000.0000,N,00000.0000,E,0,00,,,M,,,,0000*04
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,000937.799,V,0000.0000,N,00000.0000,E,,,060180,,*11
$GPVTG,,T,,M,,N,,K*4E
$GPGGA,000938.799,0000.0000,N,00000.0000,E,0,00,,,M,,,,0000*0B
...

$ picocom -b 115200 /dev/ttyACM0
picocom v1.4

port is        : /dev/ttyACM0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
noinit is      : no
noreset is     : no
```

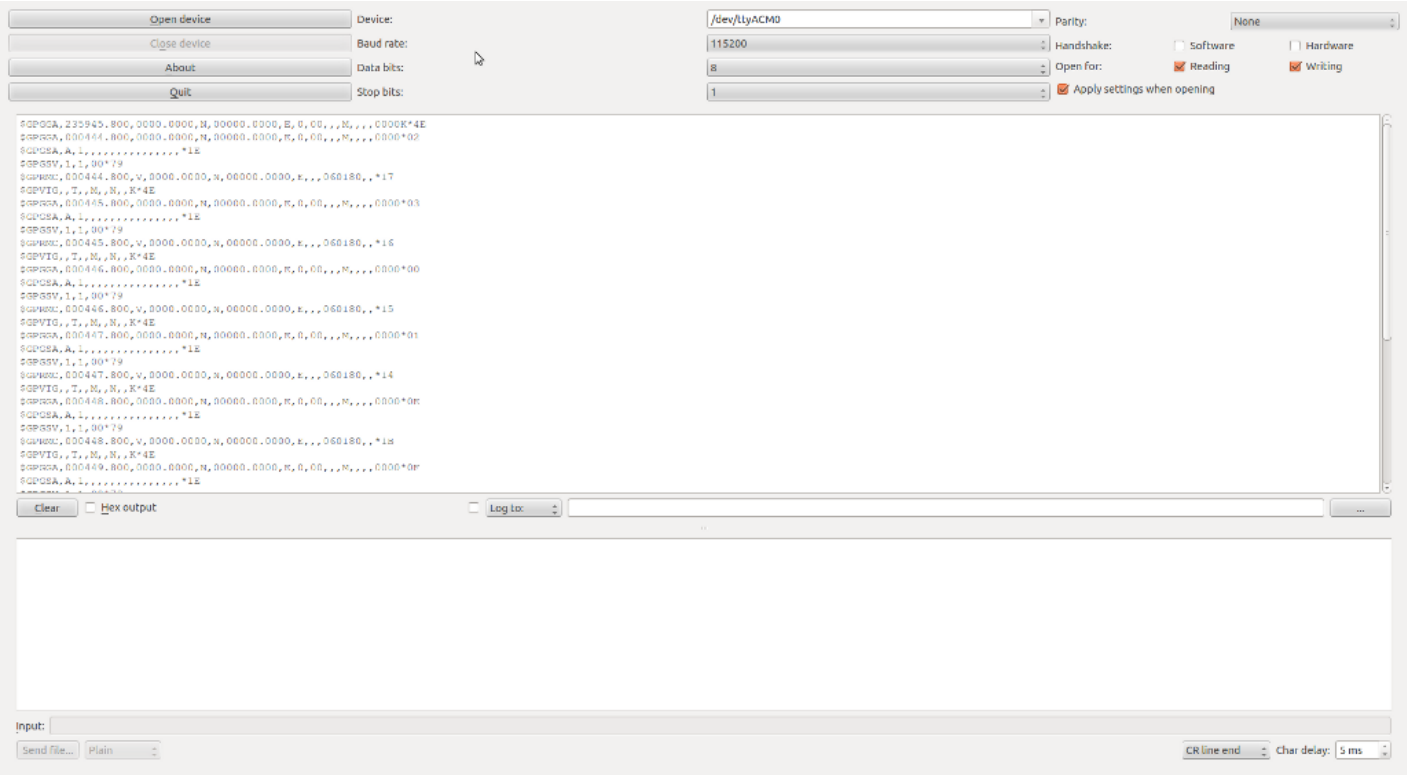


```
nolock is      : no
send_cmd is    : ascii_xfr -s -v -l10
receive_cmd is : rz -vv

Terminal ready
$GPGGA,064509.842,4405.2015,N,00457.8701,E,0,00,,,M,,,,0000*05
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPGSV,3,1,12,29,80,028,,31,61,278,,25,55,092,,21,29,178,*73
$GPGSV,3,2,12,26,28,296,,12,17,102,,02,14,041,,14,06,218,*74
$GPGSV,3,3,12,05,06,076,,20,03,123,,23,02,333,,16,02,290,*77
$GPRMC,064509.842,V,4405.2015,N,00457.8701,E,,,041015,,*1E
$GPVTG,,T,,M,,N,,K*4E
...
```

Remarque : 115200 est le débit de la "ligne" en bits/s.

Ou avec le logiciel `cutecom` :



Traitement

Lire la mise en oeuvre d'un port série sous Qt [PDF].

Les phrases NMEA 183 devront être réceptionnées puis traitées par l'application. Celles-ci étant composées de caractères ASCII, cela revient à traitert des chaînes de caractères.

Attention : la tâche d'acquisition de phrases NMEA 183 doit s'assurer de fournir des phrases "complètes". C'est-à-dire qu'elles commencent par le délimiteur '$' et se terminent par un '\r'.

Sous Qt, la classe `QString` contient de nombreuses méthodes pour manipuler des chaînes de caractère. Il vous faudra consulter très souvent sa documentation : doc.qt.io/qt-4.8/qstring.html.

Voici quelques exemples d'utilisation de la classe `QString` dans le cadre d'un **traitement d'une trame NMEA 0183** :

```
#include <QDebug>
#include <QString>

int main(int argc, char *argv[])
{
```



```
    QString phrase = "$GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E";
    // Faire des essais :
    //QString phrase = "";
    //QString phrase = "GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E";
    //QString phrase = "$GPAAM,A,A,0.10,N,WPTNME*32";
    //QString phrase = "$GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000";

    QString checksum;
    const QString debutTrame = "$";
    const QString typeTrame = "GPGGA";
    const QString debutChecksum = "*";

    // phrase vide ?
    if(phrase.length() != 0)
    {
        // est-ce une phrase NMEA 0183 ?
        if(phrase.startsWith(debutTrame))
        {
            // est-ce la bonne phrase ?
            if(phrase.startsWith(debutTrame + typeTrame))
            {
                // y-a-t-il un checksum ?
                if(phrase.contains(debutChecksum))
                {
                    checksum = phrase.section(debutChecksum, 1, 1);
                    qDebug() << "checksum : 0x" << checksum;
                }
                else
                    qDebug() << "Attention : il n'y a pas de checksum dans cette phrase !";
            }
            else
                qDebug() << "Erreur : ce n'est pas une trame GGA !";
        }
        else
            qDebug() << "Erreur : ce n'est pas une trame NMEA 0183 !";
    }
    else
        qDebug() << "Erreur : phrase vide !";

    return 0;
}
```

L'autre utilisation fréquente de la classe `QString` est la **conversion des données numériques** (`int`, `double`, ...) :

~ {.cpp} #include #include

int main(int argc, char *argv[]) { / Exemple de base */ int i = 2; double d = 3.14;

```
// Du numérique -> chaîne de caractères
QString entier = QString::number(i); // int -> QString
QString reel = QString::number(d); // double -> QString

qDebug() << "L'entier i : " << entier;
qDebug() << "Le réel d : " << reel;

// De chaîne de caractères -> numérique
entier = "100";
reel = "2.71828";
i = entier.toInt(); // QString -> int
d = reel.toDouble(); // QString -> double

qDebug() << "L'entier i : " << i;
qDebug() << "Le réel d : " << d;

/* Exemple appliqué */
QString phrase = "$GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E";
QString horodatage;
```



```
unsigned int heure, minute;
double seconde;

// découpe la trame avec le délimiteur ',' et récupère le deuxième champ
horodatage = phrase.section(',', 1, 1);
// découpe une chaine à partir d'une position et un nombre de caractères
heure = horodatage.mid(0, 2).toInt();
minute = horodatage.mid(2, 2).toInt();
seconde = horodatage.mid(4, 2).toDouble();

qDebug() << "Horodatage : " << horodatage;

horodatage = QString::number(heure) + " h " + QString::number(minute)
             + " " + QString::number(seconde) + " s";

qDebug() << "Horodatage : " << horodatage;

return 0;
```

} ~ {.cpp}

On doit maintenant s'assurer d'extraire des phrases "complètes" des données reçues du récepteur GPS. Une phrase "complète" commence par le délimiteur '$' et se terminent par un '\r' :

```
void extrairePhrases(const QString &donneesRecues)
{
    QString phrase = "";
    bool debutPhrase = false;
    bool finPhrase = false;

    for(int i = 0; i < donneesRecues.length(); i++)
    {
        // début d'une phrase NMEA183 ?
        if(donneesRecues.at(i) == '$')
        {
            debutPhrase = true;
        }
        // fin d'une phrase NMEA183 ?
        if(donneesRecues.at(i) == '\r' && debutPhrase == true)
        {
            finPhrase = true;
        }
        if(debutPhrase == true && finPhrase == false)
        {
            phrase += donneesRecues.at(i);
        }
        if(finPhrase == true)
        {
            qDebug() << "phrase NMEA183 : " << phrase;

            // on continue ...
            phrase.clear();
            finPhrase = false;
            debutPhrase = false;
        }
    }
}

int main()
{
    QString donneesRecues;

    // Simule des données reçues
    donneesRecues = "GGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E\r$GPGSV
    ,3,1,12,29,80,028,,31,61,278,,25,55,092,,21,29,178,*73\r$GPRMC,064509.842,V,4405.2015,N,00457.8701,E
    ,,,041015,,*1E\r$GPVTG,,T,,M";
```



```
    extrairePhrases(donneesRecues);

    return 0;
}
```

Code source : test-mo-nmea.zip

Code source : test-mo-port.zip

Simulateur

– GPS

Retour au sommaire

