

TP N°3 : JAVA

(Développement d'une IHM pour la gestion du port série d'un GPS)

TP N°3 :Développement d'une IHM pour la gestion du port série

1. But

Mettre en place une interface homme machine sous JAVA afin d'exploiter sur un port série les trames GPS au format NMEA183

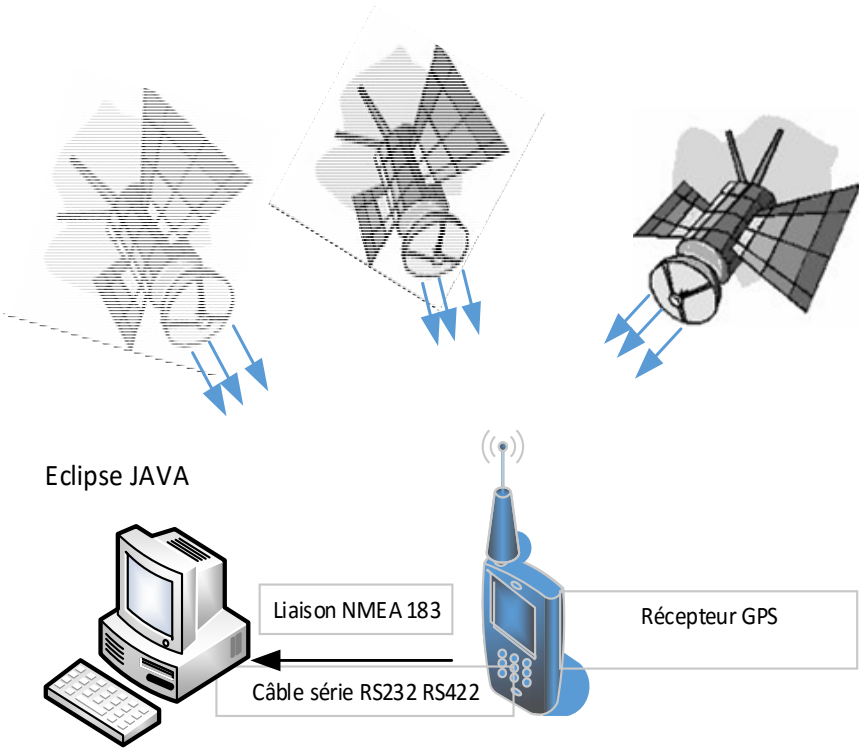
2. Objectifs : L'étudiant devra être capable de :

-De Mobiliser ces connaissances en java afin de développer un applicatif permettant la gestion du port série

3. Prérequis

- Cours sur les transmissions séries
- Cours sur le langage JAVA.

4. Schéma de câblage



5. Présentation

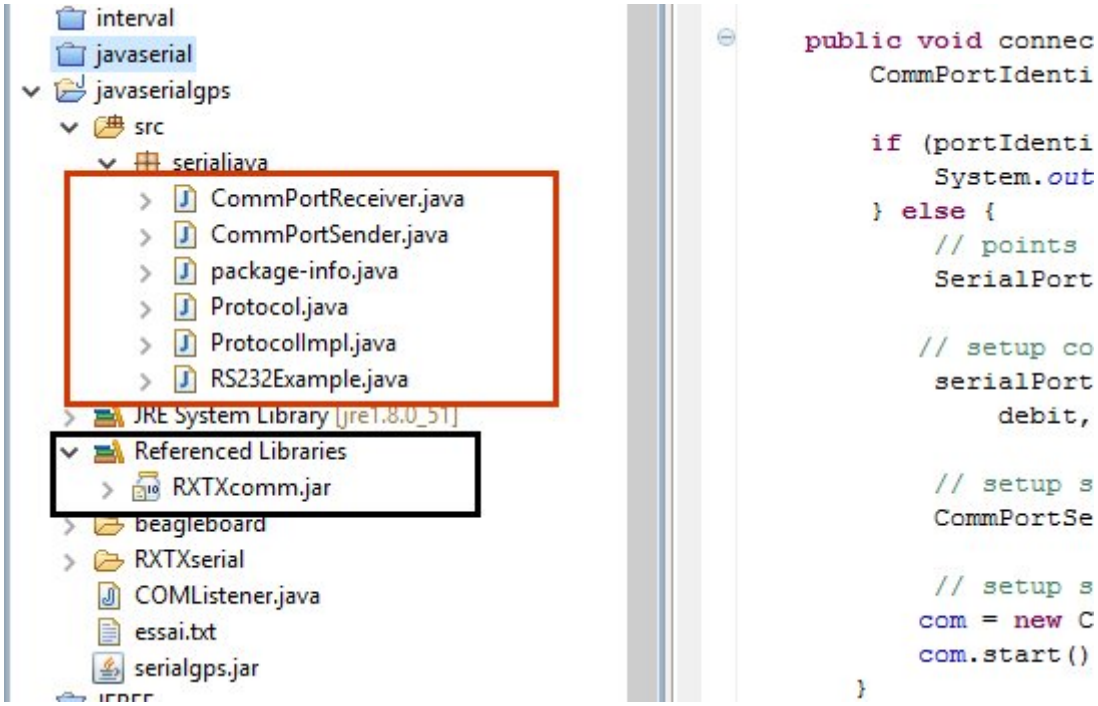
La navigation par GPS est de plus en plus importante dans le domaine maritime. On se propose lors de cette manipulation de récupérer les informations issues d'un récepteur GPS. On donne ci-dessous le schéma de principe de cette liaison. Ce TP doit nous permettre de dialoguer avec un récepteur GPS en utilisant le port série. La librairie RxTXcomm.jar dans Eclipse, afin de pouvoir bénéficier des différentes méthodes, nous permettra de recueillir la trame GPS sur le port série (annexe la librairie RXTXcomm).

La première partie consiste à développer un ensemble de classe nous permettant de récupérer la trame au format NMEA183. A l'issue on archivera cet ensemble de classe dans un fichier serialgps.jar. La dernière partie nous amènera à développer une IHM sous Windowsbuilder afin d'exploiter la latitude et la longitude

6. Récupération de la trame NMEA183 sur le port série

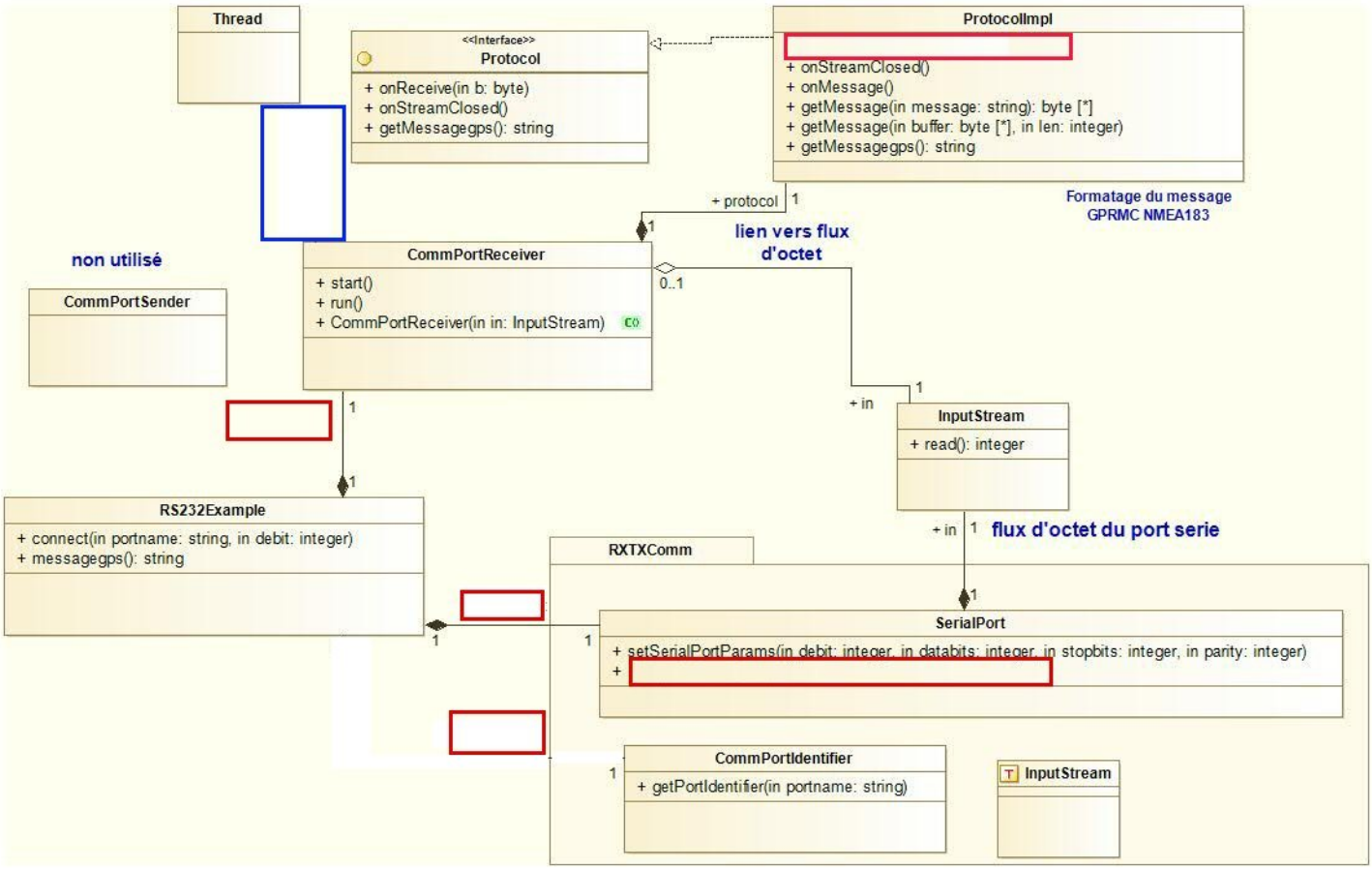
6.1) Le paquetage serialgps.jar

Ouvrir le projet javaserialgps et vérifiez la présence de l'archive RXTXcomm.jar



Cinq classes nous permettent de gérer le port série avec un protocole NMEA 183 (voir page N° pour les classes)

En vous aidant des différentes classes, complétez les éléments manquants du diagramme de classe ci-dessous :



Quel est le rôle de la méthode getInputStream

✍

Que permet de faire la méthode read() de la classe InputStream. L'appel de cette méthode est lancé dans la classe CommPortReceiver

✍

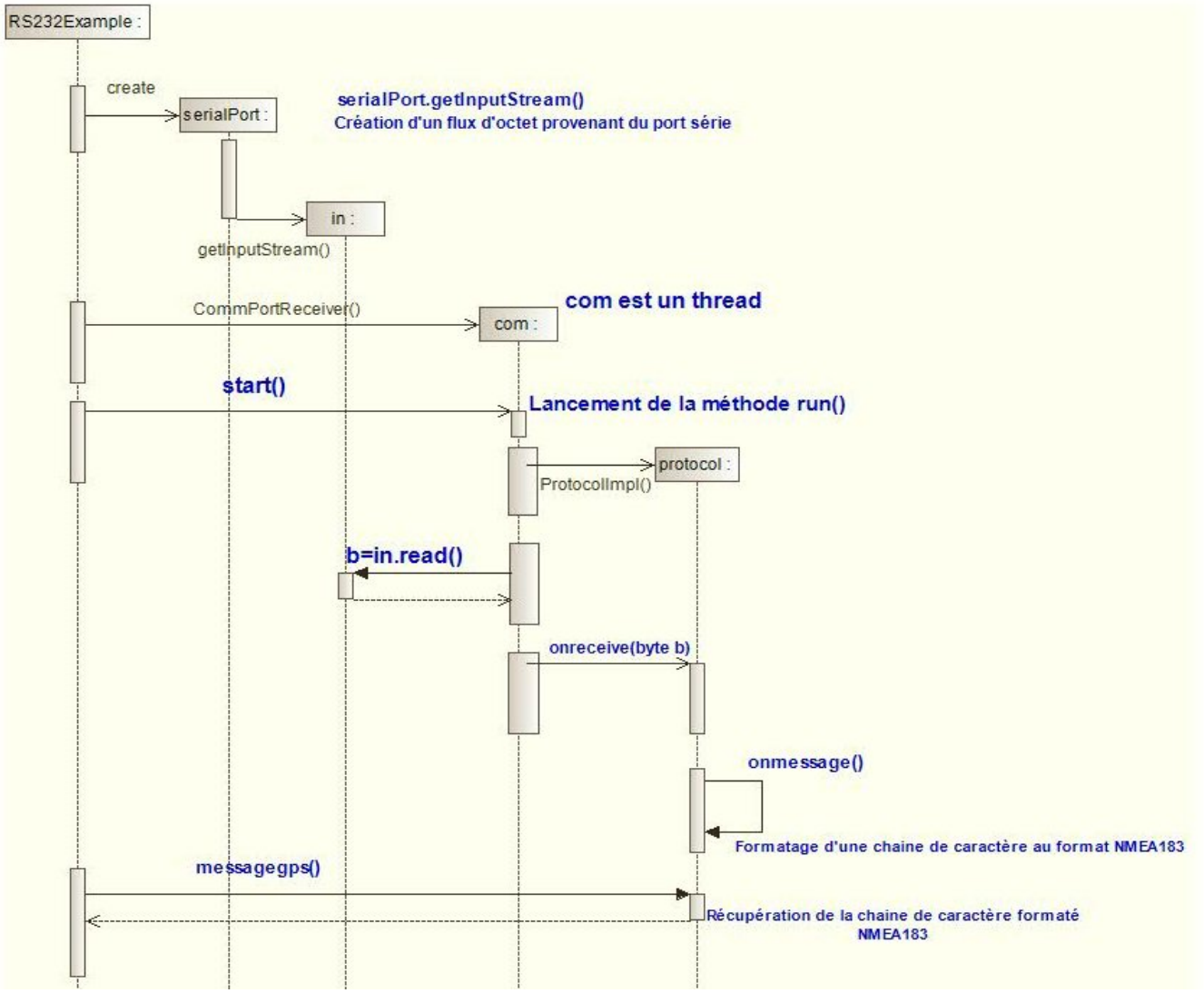
Rappelez le rôle de la méthode start() de la classe Thread

✍

Que fait le thread objet com ?

✍

Le diagramme de séquence est le suivant :



Décrire le principe de la méthode run() de l'objet com

✍

Décrire le principe de la méthode onmessage() de l'objet protocol

✍

Pour la suite de notre développement de notre applicatif on se limitera à l'utilisation de la classe RS232Example

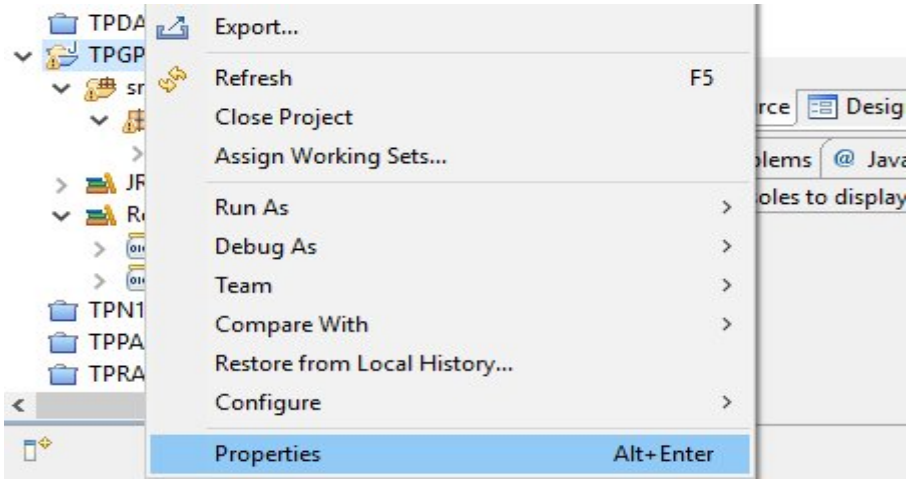
Créez un paquetage pour l'ensemble de ces classes appelé serialgps.jar (voir tutorial Eclipse)

6.2) Création de l'interface homme machine

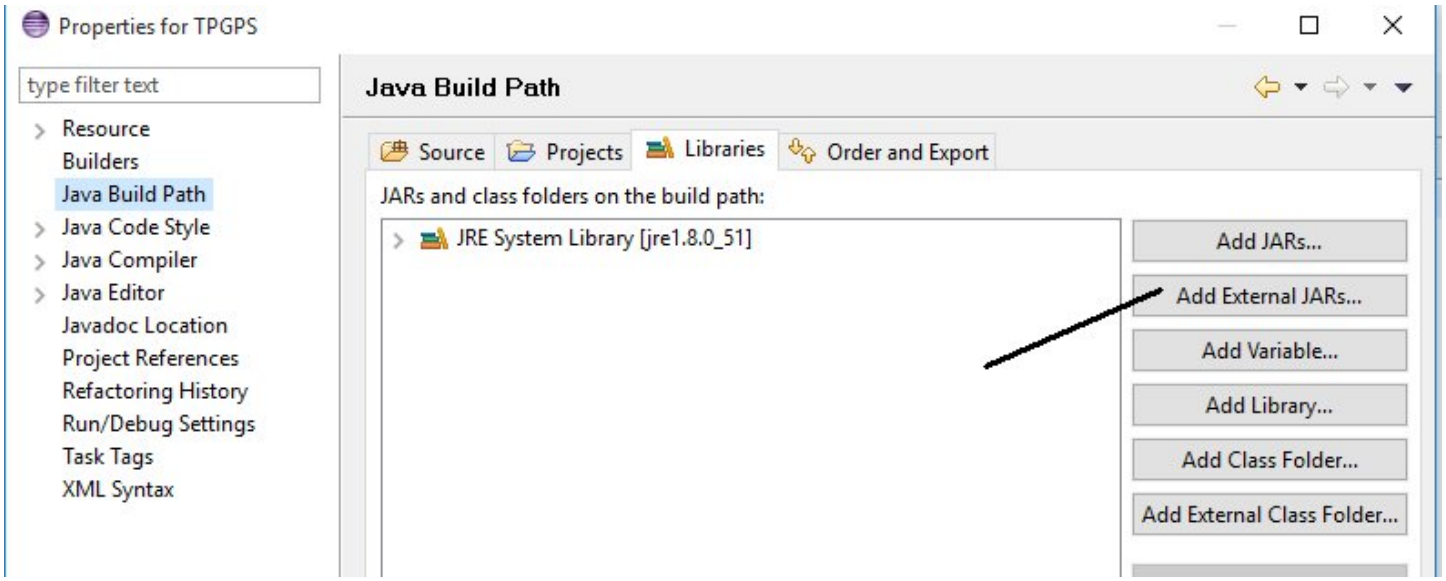
a) Créez sous Eclipse un projet nommé TPGPS

b) Par la méthodologie suivante intégrez à ce projet la librairie serialgps.jar

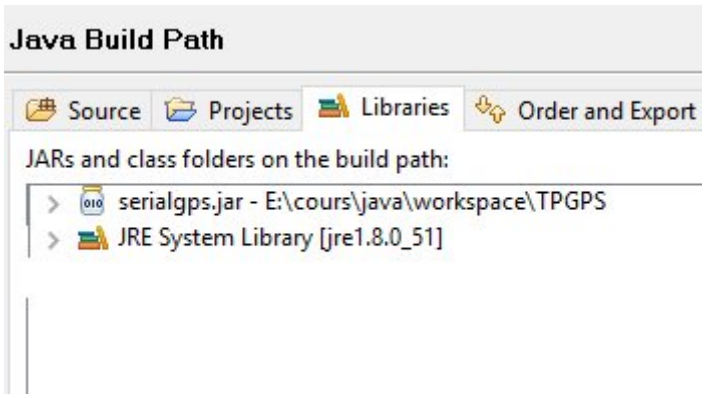
Cliquez droit sur le nom du projet et choisir properties



Sélectionnez Java Build Path et cliquez sur add External JAR

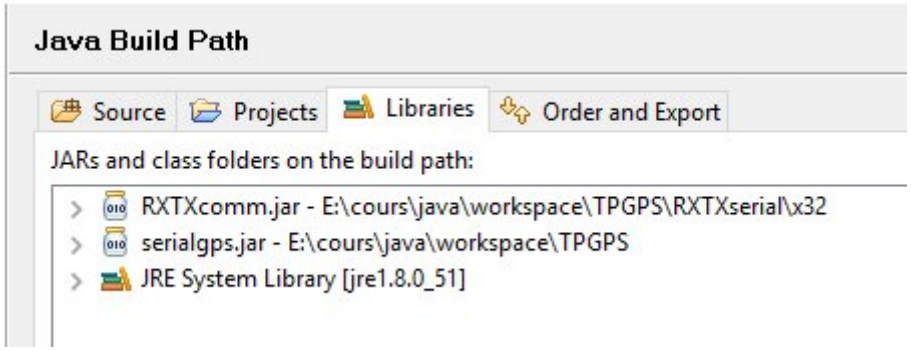


Naviguez et sélectionnez le paquetage serialgps.jar

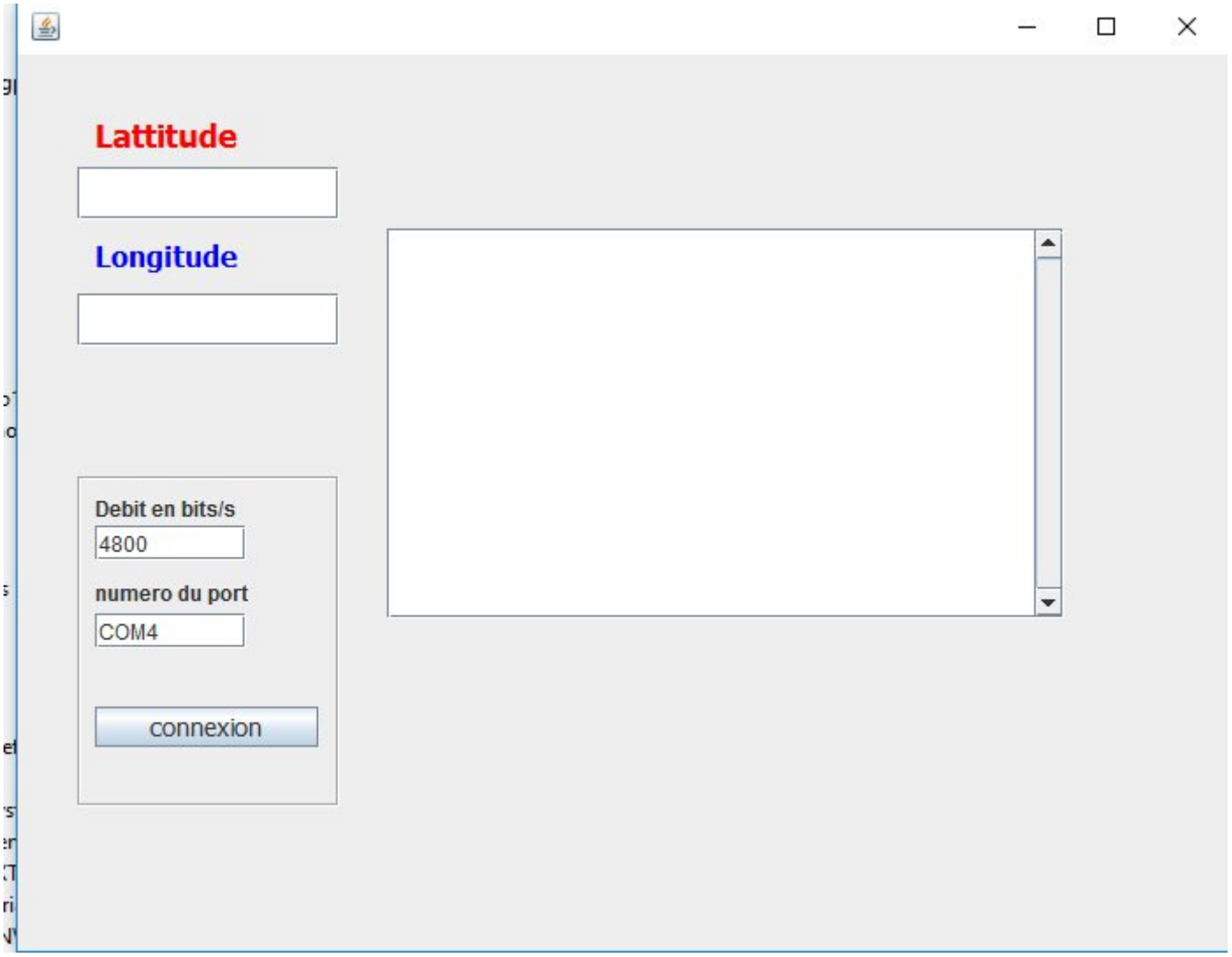


Le paquetage serialsgps.jar a besoin du paquetage RXTXComm.jar. Intégrez le paquetage RXTXComm.jar en utilisant le tutorial eclipse

Nos librairies sont prêtes :



Notre applicatif sera le suivant :



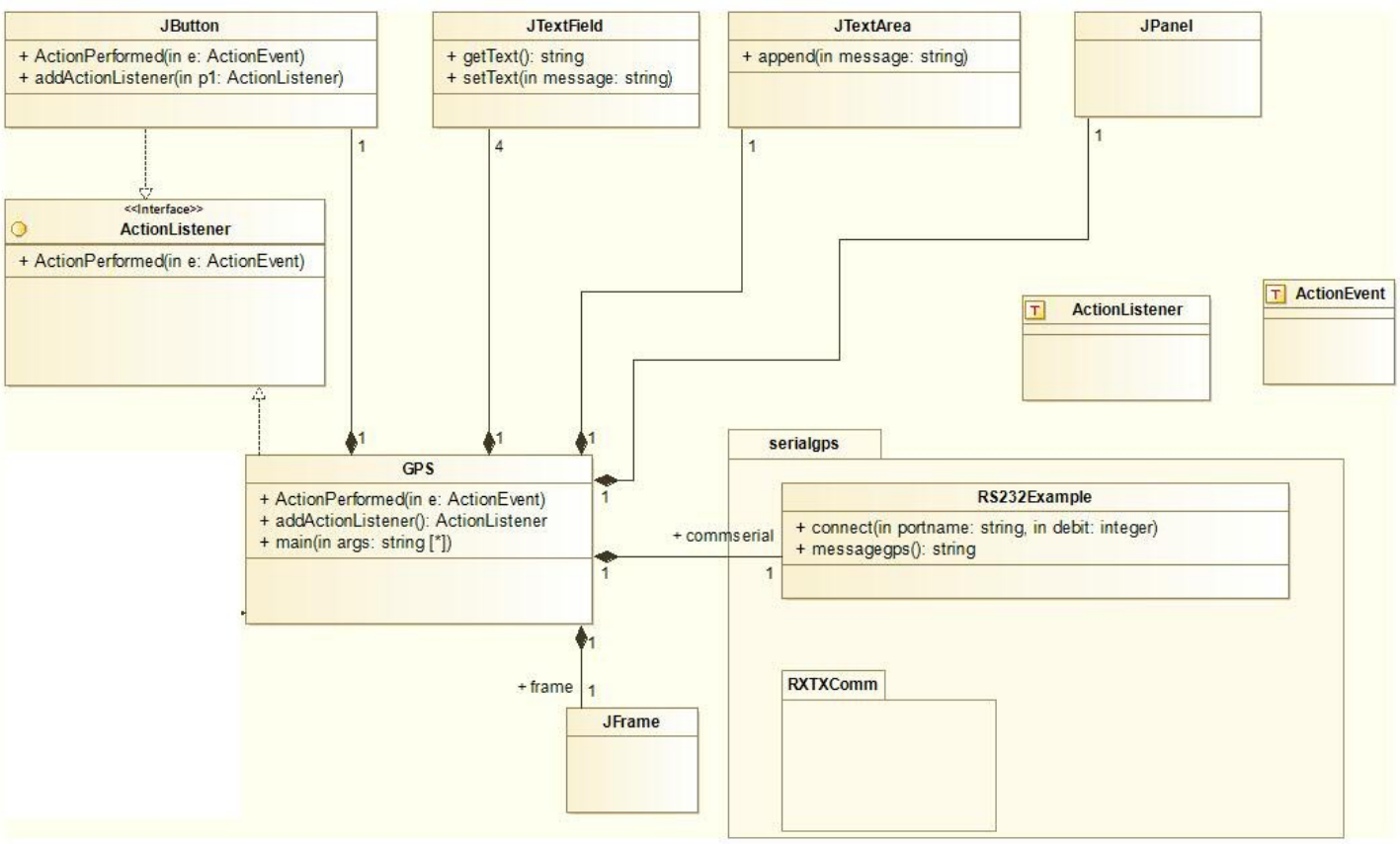
Un panneau pour saisir le port com et le débit.

Deux objets du type Jtextfield pour afficher la Latitude et la longitude

Un TextArea pour afficher les trames reçus.

Le bouton connexion nous permettra de nous connecter sur le port série physique le plus souvent le port com1

Le diagramme de classe est donné page suivante :

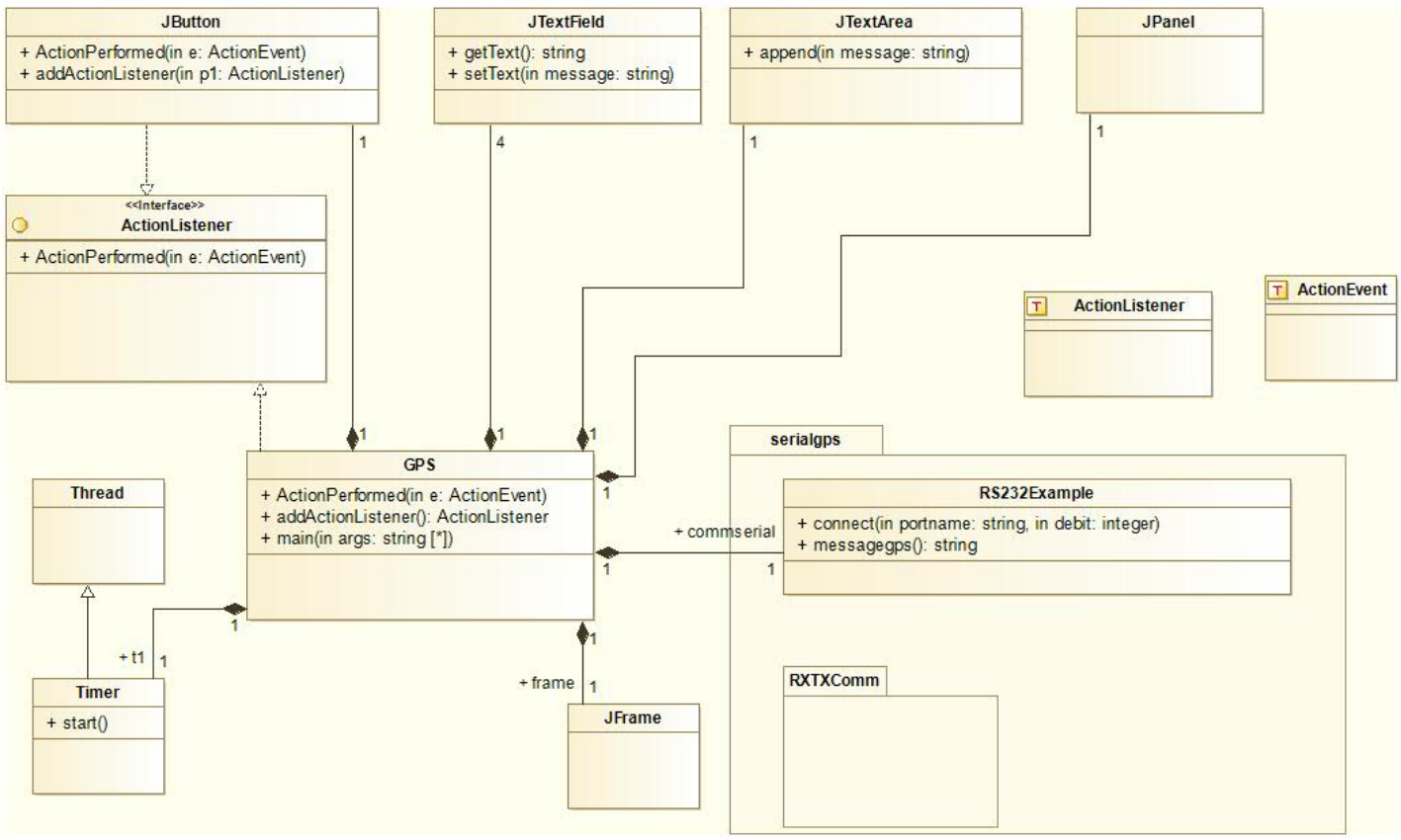


d) Réalisez sous windowsbuilder cette IHM.

Le bouton connexion nous permettra de créer un nouvel objet *commserial. Ce nouvel objet appellera sa méthode connect afin de lancer la connexion*

e) Grâce à un objet t1 de la classe **Timer** on souhaite récupérer **toutes les secondes** le message GPS (méthode messagegps()). La classe Timer est décrite dans le dossier annexe. Créez cet objet t1 dans la méthode constructeur de la classe GPS et lancez par la méthode start() ce timer

Le nouveau diagramme de classe est donné page suivante :

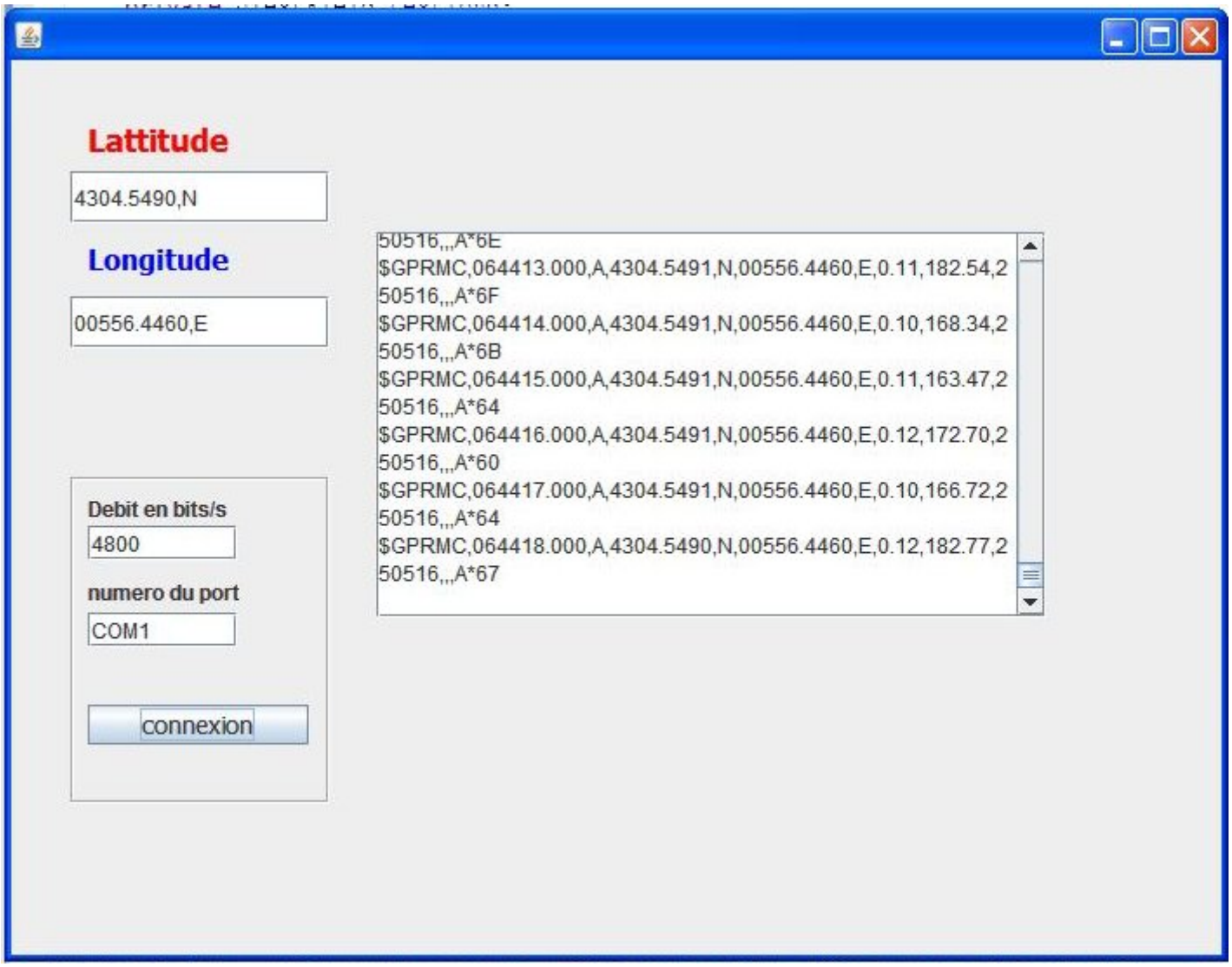


f) Complétez la méthode ActionPerformed de la classe GPS afin de :

- Acquérir le message GPS (méthode messagegps())
- Mettre à jour la zone de texte avec le message GPS
- Extraire la latitude et la longitude du message GPS (NMEA 183 annexe)
- Affichez dans les boites de texte la latitude et la longitude.

g) Branchez le GPS et vérifiez le bon fonctionnement de votre applicatif

Voir page suivante :


Lattitude
4304.5490,N
Longitude
00556.4460,E
Debit en bits/s
4800
numero du port
COM1
connexion
50516,,,A*6E
$GPRMC,064413.000,A,4304.5491,N,00556.4460,E,0.11,182.54,2
50516,,,A*6F
$GPRMC,064414.000,A,4304.5491,N,00556.4460,E,0.10,168.34,2
50516,,,A*6B
$GPRMC,064415.000,A,4304.5491,N,00556.4460,E,0.11,163.47,2
50516,,,A*64
$GPRMC,064416.000,A,4304.5491,N,00556.4460,E,0.12,172.70,2
50516,,,A*60
$GPRMC,064417.000,A,4304.5491,N,00556.4460,E,0.10,166.72,2
50516,,,A*64
$GPRMC,064418.000,A,4304.5490,N,00556.4460,E,0.12,182.77,2
50516,,,A*67

RS232Example.java

```
package serialjava;

import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

public class RS232Example {
      CommPortReceiver com;

    public void connect(String portName, int debit) throws Exception {
        CommPortIdentifier portIdentifier =
CommPortIdentifier.getPortIdentifier(portName);

        if (portIdentifier.isCurrentlyOwned()) {
            System.out.println("Port in use!");
        } else {
            // points who owns the port and connection timeout
            SerialPort serialPort = (SerialPort) portIdentifier.open("RS232Example",
2000);

           // setup connection parameters
            serialPort.setSerialPortParams(
                debit, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);

            // setup serial port writer
            //CommPortSender.setWriterStream(serialPort.getOutputStream());

            // setup serial port reader
           com = new CommPortReceiver(serialPort.getInputStream());// passage du flux
d'octet du port serie vers l'objet commportreceiver
           com.start();
       }


   }
    public String messagegps()
    {
      return com.protocol.getMessagegps();
    }

      }
```

Protocol.java

```
package serialjava;
public interface Protocol {

    // protocol manager handles each received byte
    void onReceive(byte b);

    // protocol manager handles broken stream
    void onStreamClosed();

    public String getMessagegps();

}
```

ProtocolImpl.java

```
package serialjava;

public class ProtocolImpl implements Protocol {
    byte[] buffer = new byte[1024];
    int tail = 0;
    String message1;
    String message;
    public void onReceive(byte b) {
        // simple protocol: each message ends with new line
       if (b==0x0A) {  // si octet recu = fin de ligne alors lance onMessage();
            onMessage();
        } else {  // sinon continue à receptionner et memorise dans tableau buffer b
            buffer[tail] = b;
            tail++;
        }
    }

    public void onStreamClosed() {
        onMessage();
    }



    private void onMessage() {
        if (tail!=0) {
      // constructing message
       message1 = getMessage(buffer, tail); // reprendre la chaine ou on sait arrêter
           // System.out.println("RECEIVED MESSAGE: " + message1);
            String GPRMC =message1.substring(0,6);
         if (GPRMC.equals("$GPRMC")){message=message1.substring(0, message1.length()-1);
// extraire que les trames types GPRMC
        System.out.println("RECEIVED MESSAGE: " + message);}

            tail = 0;
        }
    }

   // helper methods
   public byte[] getMessage(String message) {
        return (message+"\n").getBytes();
   }

   public String getMessagegps(){

                    return  message;
   }


    public String getMessage(byte[] buffer, int len) {
        return new String(buffer, 0, tail);
   }
}
```

CommPortReceiver.java

```
package serialjava;

import java.io.IOException;
import java.io.InputStream;


public class CommPortReceiver extends Thread {

    InputStream in;
   Protocol protocol = new ProtocolImpl();

    public CommPortReceiver(InputStream in) {
        this.in = in;  // copie du flux d'entree du port série dans la classe
    }

    public void run() {
        try {
            int b;
            while(true) {


                while((b = in.read()) != -1) {  // lecture d'un octet sur le port série
                    protocol.onReceive((byte) b);  // formatage d'une chaine type GPRMC
                }


                // wait 10ms when stream is broken and check again
                sleep(10);
            }
        } catch (IOException e) {
           e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
       }
    }
}
```

Annexe : Le standard NMEA183

1.1) Son format

Ce sont des séquences de lettres et de chiffres codés sous la forme de caractère ASCII imprimable. Une séquence se compose :

- d'un en-tête de longueur fixe (5 lettres)
- précédé d'un préfixe (un symbole, le $ en principe).

L'en tête explique au récepteur le sens des informations qui le suivent. La longueur du texte suivant l'en tête est variable.

La transmission de ces chaines de caractère ASCII s'effectue grâce à une liaison série du type asynchrone, dont les caractéristiques sont les suivantes :

- 4800 bauds ou bits/s
- 8 bits par caractères
- pas de parité
- 1 bit de start
- 1 bit de stop

La norme électrique utilisée pour véhiculer ces informations est le plus souvent la norme RS422 voir la norme RS232.

1.2) Les messages de diffusion

Format simplifié des trames NMEA :

$	tt	<id>	<données>	*<checksum>	CR	LF

- Le champ tt indique la nature de l'information
- <id> est une suite de 3 caractères (GGA, GLL, GSV, ...) définissant le type de la trame
- les données sont séparées par des virgules, leur nombre dépend du type de trame
- la somme de contrôle (checksum) est précédée d'un caractère '*', elle est calculée par OU exclusif entre tous les caractères situés entre '$' et '*' . Ce champ est présenté sous forme hexadécimale, sa présence est en général facultative.
- Une trame NMEA ne comporte jamais de caractère Espace (0x20).

Description du champ tt :

AG = Pilote automatique (cas général)
AP = Pilote automatique (magnétique)
CC = Ordinateur
CD = Appel sélectif (radio)
CS = Communication par satellite

CT = Radio téléphone MF/HF
CV = Radio téléphone VHF
CX = Récepteur à scanner
DE = DECCA
DF = « direction finder »
EC = Cartographie électronique et système d'information
EP = Balise de position de détresse
ER = Système de contrôle de la salle des machines
GP = GPS
HC = Compas
HE = Gyroscope « North Seeking »
HN = Gyroscope « Non-North Seeking »
II = Instrument intégré
IN = Instrument de navigation intégré
LA = Loran A
LC = Loran C
OM = Oméga
RA =Radar
SD = Sondeur
TR = Système de positionnement
SS = Sondeur à scanner
TI = « Turn Rate Indicator «
TR = Système de navigation TRANSIT
VD = Capteur de vitesse à effet doppler
VM = Capteur de vitesse magnétic
VW = Capteur de vitesse mécanique
YX = Transducteur
ZA = Horloge atomique
ZC = Chronomètre
ZQ = Quartz
ZV = Mise à l'heure radio
WI = Station météo

1.3) Les messages d'interrogation

Le format est : *$ttllQ,sss,[CR][LF]*
tt est l'identificateur du demandeur
ll est l'identificateur de l'émetteur requis

Le 5ème caractère est un Q
sss contient le mnémonique du message demandé.

Exemple : $CCGPQ,GGA[CR][LF] Où l'appareil « CC » (ordinateur) demande à l'appareil « GP » (GPS) le message « GCA ». Le GPS enverra alors ce message toute les seconde.

1.4) Les messages propriétaires

Le format est : *$PmmmA,par1,par2,...,[CR][LF]*
Le premier caractère est un P
mmm est le code fabriquant
GRM = Garmin
ML2 ou ML3 = MLR
MGN = Magellan
MOT = Motorola
RWI = Rockwell
SLI = Starlink

Le 5ème caractère définit le type de message (A à Z)
par1, par2, parX sont les paramètres qui dépendent du message.

1.5) Caractéristique détaillée des messages de diffusion

$GPGGA, HHMMSS, XXXX.XXX, N, XXXXX.XXX, W, X, XX, XX, XXXX, M, XXXX, M, SSSS, XXXX CR LF

HHMMSS : Heures, minutes, secondes UTC
XXXX.XXX : Latitude
N : N = Nord, S = Sud
XXXXX.XXX : Longitude
W : E = Est, W = Ouest
X : 0 = point non calé, 1 = point calé, 2 = point calé en mode différentiel, 6 point estimé

XX : Nombre de satellites
XX : Dilution horizontale de la précision
XXXX, M : Altitude en mètres de l'antenne au dessus du niveau de la mer
XXXX, M : Différence en mètres entre l'ellipsoïde WGS84 et le niveau moyen de la mer
SSSS : Age des données différentielles en secondes
XXXX : Numéro de la station différentielle
(Les données DGPS ne sont pas transmises en mode GPS normal, seules les virgules apparaissent).

*** GLL Position vraie**

$GPGLL, XXXX.XXX, X, XXXXX.XXX, X, HHMMSS, V, CR LF

XXXX.XXX : Latitude
N : N = Nord, S = Sud
XXXXX.XXX : Longitude
W : E = Est, W = Ouest
HHMMSS : Heures, minutes, secondes UTC
V : A = point valide, V = point non valide

*** GSA Dilution de la précision et satellites GPS actifs**

$GPGSA, S, D, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, X.X, X.X, X.X, CR LF

S : A = sélection automatique d'un point 2d ou 3d, M = manuel
D : 1 = point non valide, 2 = 2d, 3 = 3d
XX (12 fois): Id des satellites utilisés (Nombre aléatoire, emplacements pour 12 mais certains sont vides)
X.X : Dilution de la précision de position
X.X : Dilution de la précision horizontale
X.X : Dilution de la précision verticale

*** GSV Satellites en vue**

$GPGSV, N, X, SS, XX, XXX, XX, XX, XX, XXX, XX, XX, XX, XXX, XX, XX, XX, XXX, XX, XX, CR LF

N : Nombre de messages pour lister tous les satellites
X : Numéro de ce message
SS : Nombre de satellites en vue
XX, XXX, XX, XX : Id du satellite, élevation en degrés (max. 90), azimut en degrés (depuis le nord vrai), niveau du signal (Emplacement pour 4 satellites) **(4 fois)**