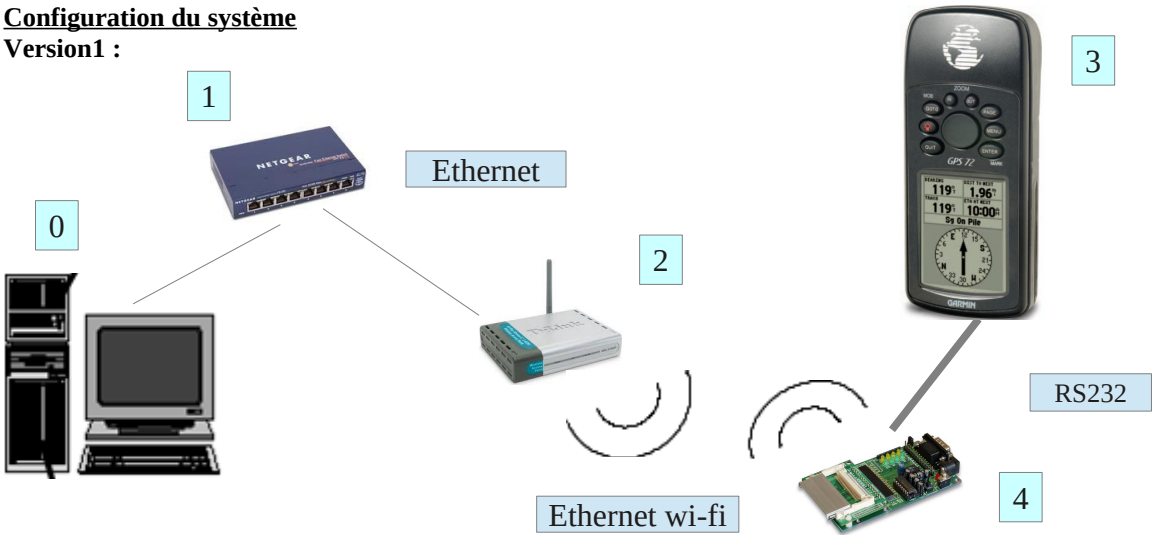
TP GPS : Application Qt & API Google maps	Durée : 8 heures
Principale compétence terminale concernée	Savoir et savoir faire associés
C4.1 Câbler et/ou intégrer un matériel C4.5 Tester et valider un module logiciel et matériel C4.6 Intégrer un module logiciel C5.3 Effectuer la recette d'un produit avec le client	SF44 Valider les fonctions du nouveau matériel SF46 Rendre opérationnel le système ou sous-système SF60 Appliquer les procédures de tests sur un équipement ou un produit et d'en déduire le bon fonctionnement SF73 Vérifier la conformité du fonctionnement.

Niveau	1ere année BTS SN IR
Configuration du poste	Possède un système windows avec accès internet + fichier de coordonnées GPS
Données fournies	Documentation API Google maps
Prérequis	IDE Qt Creator – Langage C++

Objectif :
Réaliser une application permettant de tracer un parcours avec l'API google maps, à partir de données envoyées par un GPS. Le poste de contrôle doit capturer les données du GPS, les traiter, et les représenter selon des critères choisis.

Configuration du système
Version1 :

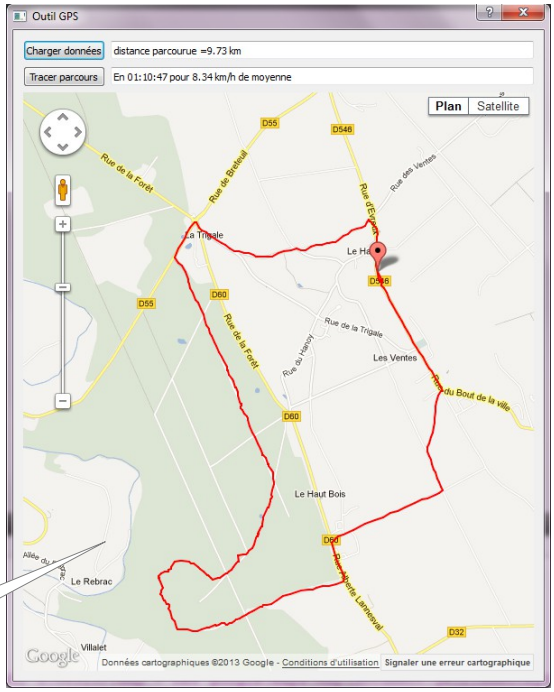


0 : Poste contrôle ; 1 : Switch ; 2 : Point d'accès wi-fi ;
3 : GPS ; 4 : Interface RS232/Wifi

Version2 :
Afin d'obtenir une variation des coordonnées de géolocalisation le synoptique du système va se résumer à un unique PC avec accès internet et un fichier .txt contenant toutes les coordonnées géographique d'un trajet.
Le fichier est constituer de la façon suivante :
…….....
t,d,48.955963,01.082992,12/23/2012,11:37:44,140.0092,0
t,d,48.955891,01.083005,12/23/2012,11:37:49,140.0092,0
t,d,48.955869,01.082985,12/23/2012,11:37:51,140.0092,0
…….....

Le séparateur de données est la virgule, données utiles :

n° champs	**description**
3	latitude
4	longitude
6	heure
7	altitude



QWebView

T.P Hutinet.

Le travail à effectuer se basera sur la configuration version2

1] Sous Qt-Creator: Création d'un projet de type boîte de dialogue

Réaliser l'IHM décrite en page 1:
Nom du projet : gps (Attention de bien choisir l'emplacement)

Vous compléterez dans le fichier .pro, la ligne suivante :
Qt += core gui **network webkit**
pour gérer la partie réseau et web dans votre programme.

2] Chargement d'une carte via google maps

Pour charger une carte dans le QWebView :
Déclarer un attribut QNetworkProxy proxy;
Copier le contenu du fichier map_html.txt dans dialog.h
Déclarer et implémenter les méthodes suivantes :

```
void Dialog::initProxy(){ // nécessaire si accès internet via proxy
    proxy.setType(QNetworkProxy::Socks5Proxy);
    proxy.setType(QNetworkProxy::HttpProxy);
    proxy.setHostName("172.16.254.254");
    proxy.setPort(3128);
    // proxy.setUser("username");
    // proxy.setPassword("password");
    QNetworkProxy::setApplicationProxy(proxy);
}

void Dialog::initMap(void){ // initialise le QWebView avec du code HTML
  QString contenu;
  // Autoriser l'exécution de code javascript
  ui->webView->settings()->setAttribute(QWebSettings::JavascriptEnabled, true );
  contenu = MAP_HTML;   // code javascript prédéfini dans MAP_HTML
  ui->webView->page()->mainFrame()->setHtml(contenu); // chargement de la page
  QTimer::singleShot(1000, this, SLOT(loading()));
}

void Dialog::loading(void){   // charge la carte dont le centre est lat,lon
  QString code = QString("latlng=new google.maps.LatLng(%1,  %2);")
                  .arg(lat).arg(lon);
  code += "map = new google.maps.Map(document.getElementById(\"map_canvas\"),
      myOptions);";
  code += "map.setZoom(15);map.setCenter(latlng);";
  ui->webView->page()->mainFrame()->evaluateJavaScript(code);
}
```

Principe : Voir doc API google maps pour les explications du contenu HTML et l'utilisation des fonctions javascript de l'API. C'est la fonction loading() qui crée les variables de position latlng et de la carte map. Elles sont déclarées dans le code HTML partie `<script  type=\"text/javascript\">` en variable globale.

En déclarant 2 attributs de la classe QDialog : qreal lat et lon et en les initialisant dans le constructeur à des coordonnées (ex : Oslo(59.9138204, 10.7387413) ; Berlin(52.5056819, 13.3232027) ; Jakarta(-6.211544, 106.845172) ; Evreux(49,0241, 1,1508)) vous afficherez une carte centrée sur ces coordonnées. Vérifier le bon fonctionnement.

3] Lecture des données du fichier : `bool Dialog::chargementDonnee(QString map)`

Méthode qui lit les données à partir du nom d'un fichier "map" (ex Map170213.txt) et remplit les listes correspondantes aux données :
Nouveaux attributs à déclarer :

```
    QList<qreal> listLat;     // liste des latitudes
    QList<qreal> listLon;     // liste des longitudes
    QList<qreal> listAlt;     // liste des altitudes
    QList<QString> listHor;   // liste des horaires
```

`T.P Hutinet.`
Le travail consiste à extraire les données de chaque ligne et les ranger dans la liste correspondante
Principe :
Tant que pas fin de fichier
QByteArray line <= lire une ligne // line= "t,d,48.955963,01.082992,12/23/2012,11:37:44,140.0092,0"
Mettre le QByteArray dans un QString //ligne="t,d,48.955963,01.082992,12/23/2012,11:37:44,140.0092,0"
Spliter le QString
ranger les valeurs dans leur tableau respectif
// 48.955963 ranger dans la liste des latitudes
// 01.082992 ranger dans la liste des longitudes
// 11:37:44 ranger dans la liste des horaires
// 140.0092 ranger dans la liste des altitudes

Types de variables utiles pour l'extraction des différentes données :
`QFile, QByteArray, QString, QStringList.`

Quelques méthodes utiles :
`setFileName(), open(), atEnd(), readLine(), split(), toDouble().`

4] Appel de la méthode "chargementDonnees : `void Dialog::on_pB_Charger_clicked()`

Sur un clic du bouton "charger données", ouvrir une boite de dialogue pour sélectionner son fichier "map" avec un objet `QFileDialog.` La méthode initMap sera maintenant appelée ici, non plus dans le constructeur. Prévoir d'initialiser les "lat" et "lon" avec la première valeur des QList associées.

5] Placer un marker : `void Dialog::placeMarker(qreal lat, qreal lon)`

En vous inspirant de la fonction loading(). Exécuter du code javascript dans la QWebView qui consiste à initialiser la variable "pos" à la première coordonnée et appeler la function javascript placeMarker().

6] Dessiner le trajet: `void Dialog::on_pB_Tracer_clicked()`

Le principe est d'initialiser les variables javascript previousPosition et currentPosition correspondantes respectivement à la première et seconde position puis d'appeler la function javascript traceParcours() qui trace une ligne entre les 2 points. Répéter cette logique jusqu'aux dernières coordonnées et vous obtiendrez le tracé du parcours.

7] Calculer la distance parcourue :

La méthode distanceParcours() retourne le nombre de km parcouru. Pour calculer la distance totale parcourue, il faudra additionner la distance entre chaque points relevés.

Le travail consiste à développer la fonction calculDistance(...). La formule est données dans les documents fournis.

```
qreal Dialog::distanceParcours(void){
    int indiceMax = listAlt.size() - 1;
    for(int i=0; i<indiceMax; i++){
        distance+= calculDistance(listLat.at(i), listLon.at(i),listLat.at(i+1),
            listLon.at(i+1), listAlt.at(i), listAlt.at(i+1));
    }
    return(distance);
}
```

8] Calculer la durée totale du trajet effectué : QString `Dialog::dureeParcours(void)`

Pour calculer ce temps, prendre les premiers et derniers éléments de listHor. Les convertir en QTime (voir doc Qt) pour effectuer les calculs et convertir à nouveau en QString pour pouvoir afficher le résultat.

Options :
Déterminer la vitesse moyenne obtenue pendant le parcours.
Placer le code javascript dans un fichier .js, séparé du fichier .html
Récupérer les coordonnées GPS sur le clic carte. (Opération inverse : Javascript => Qt)