TP QT 1

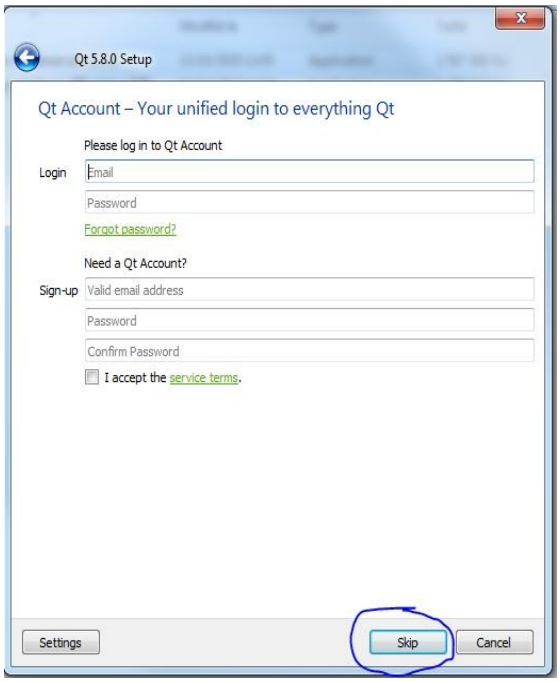
Table des matières

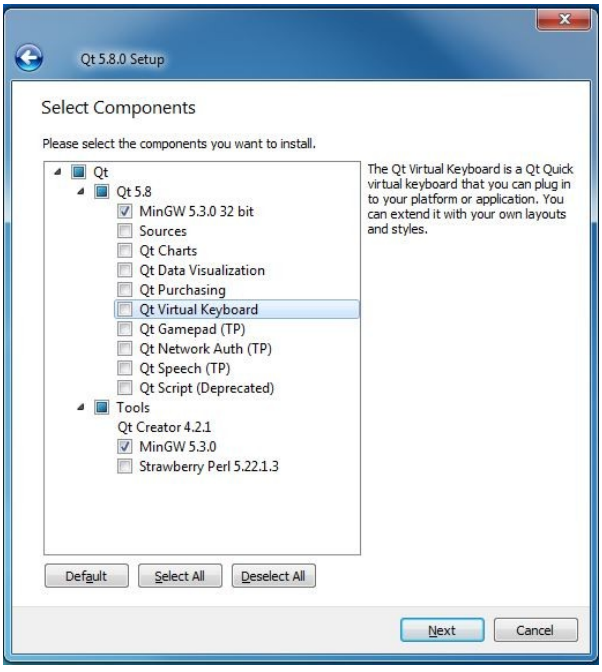
Faire un compte rendu, répondre aux questions surlignées en jaune.

- **Enregistrer le compte rendu sous CRQtTP1CPPvotreNom.pdf, rendre ce compte-rendu via Pronote avant le mardi 2 Novembre**

1 - Installation

Installer QT version 5.8.0 qui permet d'installer qt sans créer un compte QT. Copier le fichier d'installation /PartagEleve/logiciels/Qt/qt-opensource-windows-x86-mingw530-5.8.0.exe
Installer le logiciel Qt dans le répertoire par défaut C:/Qt.
Choisir skip sur le login Qt .
Sélectionner dans « select Components » seulement Mingw5,3,0 et Qtcreator MinGW 5,3,0



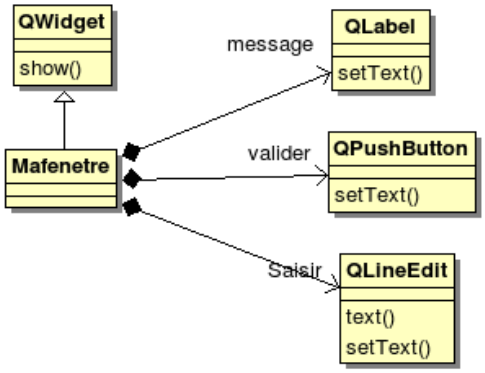


Faire un raccourci sur votre bureau de Qt Creator qui se trouve sur C:/Qt/Qt5.8.0/tools/QtCreataor/bin/qtcreator.exe

Ce site http://qtmirror.ics.com/pub/qtproject/official_releases/qt/5.8/5.8.0/ semble fournir toutes les plateformes de la versions 5,8,0

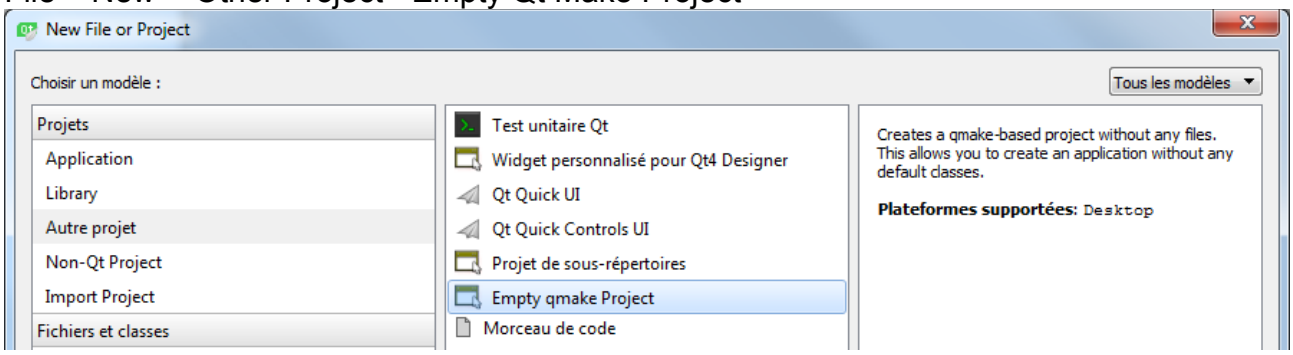
2 – UML du projet

On va utiliser les classes QWidget,QLabel, QPushButton, QLineEdit de Qt .

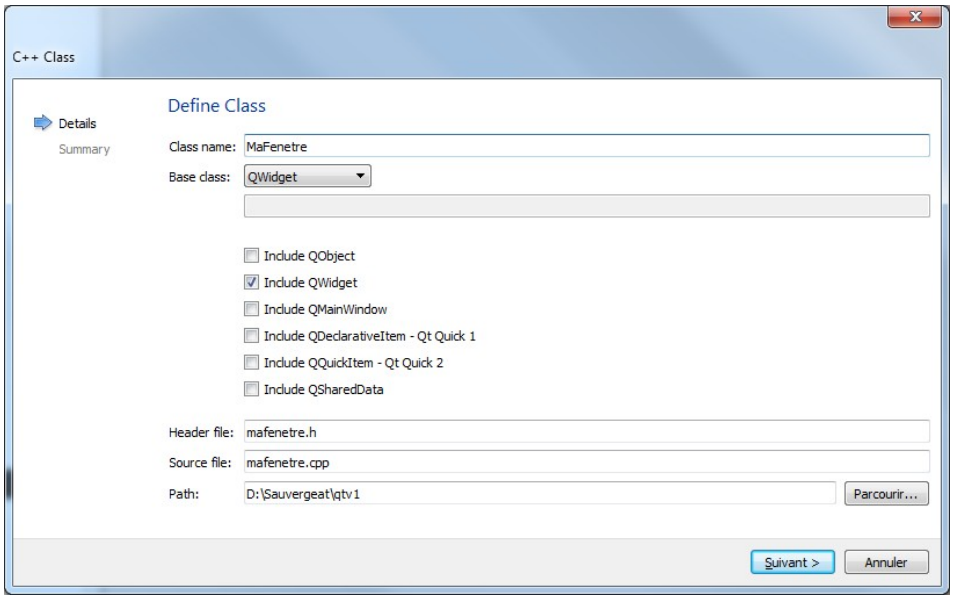


3 Étapes de création

Lancer le programme QtCreator.
File – New- -Other Project - Empty Qt Make Project



File - New – C++ Class - Class Name « MaFenetre » Base Class « QWidget »



File - New – C++ Source - « main »

3.1 - le main

Le Main crée un environnement permettant d'afficher le contenue de la classe Mafenetre.

```
#include <QApplication>
#include "mafenetre.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MaFenetre w;
    w.show();

    return a.exec();
}
```

Ajouter les lignes dans le fichier **.pro**

```
QT  += core gui
QT += widgets

TARGET = test
TEMPLATE = app
```

Lancer le programme.
Il est vide . **Logique …**

3.2 - la Classe MaFenetre : l'entête .h

Ajout d'une Librairie :

```
#include <QtWidgets>
```

Dans la classe, ajouter ces attributs :

```
private :
      QLabel * message;
      QLineEdit * saisir;
      QPushButton  * valider;
```

3.3 - la Classe MaFenetre : la définition de la classe .cpp

Dans le Constructeur : faire l'allocation dynamique

```
    message = new QLabel;
   saisir= new QLineEdit;
   valider= new QPushButton;
```

Dans le Constructeur ajouter ces 3 lignes :

```
    message->setText("message a modifier");
   saisir->setText("Entrez du texte !");
   valider->setText("VALIDER");
```

Pour gérer la disposition des objets de façon dynamique, nous allons utiliser les Layouts. La librairie Qt offre un dispositif qui simplifie considérablement le positionnement des objets (widgets) dans un dialogue lorsqu'on ne souhaite pas obtenir des effets très particuliers.
Si l'on dispose d'un QHBoxLayout, les widgets qu'on y insérera seront placés les uns à côté des autres. Dans le cas d'un QVBoxLayout, les widgets insérés prendront au contraire place les uns en dessous des autres.

```
   // Définir un layout vertical permettant de positionner des objets
    QVBoxLayout * vertical = new QVBoxLayout(this);
   vertical->addWidget(message);
   vertical->addWidget(saisir);
   vertical->addWidget(valider);
```

Compilez et lancez le programme. Le bouton n'a pas d'action. **Logique.**

3.4 - la gestion d'action utilisateur

Chaque action de l'utilisateur est géré comme un signal. Le signal peut être connecté ou pas à l'appel d'une méthode.
On va connecter l' action (SIGNAL) de l'utilisateur , vers un appel de méthodes (SLOT).
// Ajouter dans Mafenetre.h

```
public slots:
    void f_valider();
```

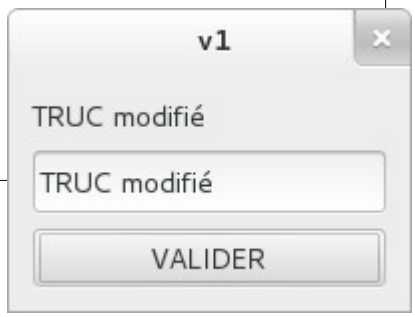
// Ajouter dans le constructeur

```
   // associe l'action de clicker sur un bouton à l'appel de fonction f_valider
    connect(valider, SIGNAL(clicked()), this, SLOT(f_valider()));
```

//Code de la fonction

```
void MaFenetre::f_valider()
{
    QString m;
  // Fenêtre Message
   QMessageBox::information(0, "Qtv1", "Modification du Label");
    // Récupérer le mot saisie dans le QLineEdit
    m=saisir->text();
    // Modifier le label
    message->setText(m);

}
```



Lancer le programme et tester l'action du bouton.

Compte - Rendu :
Mettre une copie d'écran de l'IHM.

4 Premier exercice : Convertisseur Kms → Miles (6 pts)

L'objectif de ce programme est :

Lancer le programme QtCreator.
File – New- -Other Project -Empty Qmake Project
File - New – C++ Class - Class Name « ConvKM » Base Class « QWidget »
File - New – C++ Source - « main »

4.1 - le main

Le Main crée un environnement permettant d'afficher le contenu de la classe ConvKm.

```
#include <QApplication>
#include "convkm.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    ConvKM w;
    w.show();
    return a.exec();
}
```

Ajouter les Lignes dans le .pro

```
QT       += core gui

QT += widgets

TARGET = testComKM
TEMPLATE = app
```

4.2 - la Classe ConvKM : l'entête .h

Ajout d'une Librairie :

```
#include <QtWidgets>
```

Dans la classe, ajouter ces attributs :

```
private :
     QLineEdit * lineEditKm1;
     QLineEdit * lineEditKm2;
     QLineEdit * lineEditMilles1;
     QLineEdit * lineEditMilles2;
     QLabel * labelTitre;
```

```
    QLabel * labelKm;
    QLabel * labelMilles;
    QLabel * labelConv1;
    QLabel * labelConv2;
    QRadioButton * radioKmM;
    QRadioButton * radioMKM;
    QGridLayout* layGeneral;
```

4.3 - la Classe ConvKM : la définition de la classe .cpp

Dans le Constructeur ajouter ces lignes :

```
// QTextCodec::setCodecForCStrings(QTextCodec::codecForName("UTF-8"));
  lineEditKm1=new QLineEdit(this);
  lineEditKm2=new QLineEdit(this);
  lineEditMilles1=new QLineEdit(this);
  lineEditMilles2=new QLineEdit(this);
  labelTitre= new QLabel(this);
  labelTitre->setText("Convertisseur Kilomètres en Milles");
  QFont fontTitre( "Arial", 20, QFont::Bold);
  QPalette pal(0x80ffff); // définition d'une couleur
  labelTitre->setAutoFillBackground(true);
  labelTitre->setPalette(pal); // affecte la couleur de font au Label
  labelTitre->setFont(fontTitre);
  labelTitre->setAlignment(Qt::AlignHCenter);
  labelKm=new QLabel(this);
  labelKm->setText("Kilomètres");
  labelMilles=new QLabel(this);
  labelMilles->setText("Milles");
  labelConv1=new QLabel(this);
  labelConv1->setText("Km->Milles");
  labelConv2=new QLabel(this);
  labelConv2->setText("Km<-Milles");
  radioKmM=new QRadioButton(this);
  radioKmM->setText("Km->Milles");

  radioMKM=new QRadioButton(this);
  radioMKM->setText("Km<-Milles");
  QValidator* validator = new QDoubleValidator();
  lineEditKm1->setValidator(validator);
  lineEditKm2->setEnabled(false);
  QValidator* validator2 = new QDoubleValidator();
  lineEditMilles2->setValidator(validator2);
  lineEditMilles1->setEnabled(false);
  lineEditMilles2->setEnabled(false);
  radioKmM->setChecked(true);

  this->setWindowTitle("Convertisseur Milles Kilometres");
```

Pour gérer la disposition des objets de façon dynamique, nous allons utiliser le Layouts. Gridlayout

Compte - Rendu :

Expliquez la différence entre QgridLayout et QVBoxlayout. :http://doc.qt.io/qt-4.8/layout.html

Expliquez la différence entre :

```
layGeneral->addWidget(labelTitre,1,1,1,4);
layGeneral->addWidget(lineEditKm1,3,1);
```

Expliquez l'intérêt d'hériter de Qwidget dans la classe Convkm

```
   // Définir un layout, sous forme d'une grille
    layGeneral = new QGridLayout(this);

    layGeneral->addWidget(labelTitre,1,1,1,4);
    layGeneral->addWidget(lineEditKm1,3,1);
    layGeneral->addWidget(lineEditKm2,4,1);
    layGeneral->addWidget(lineEditMilles1,3,3);
    layGeneral->addWidget(lineEditMilles2,4,3);
    layGeneral->addWidget(labelKm,2,1);
    layGeneral->addWidget(labelMilles,2,3);
    layGeneral->addWidget(labelConv1,3,2);
    layGeneral->addWidget(labelConv2,4,2);
    layGeneral->addWidget(radioKmM,3,4);
    layGeneral->addWidget(radioMKM,4,4);
```

4.4 - la gestion d'action utilisateur

Chaque action de l'utilisateur est géré comme un signal. Le signal peut être connecté ou pas à l'appel d'une méthode.
On va connecter l' action (SIGNAL) de l'utilisateur , vers un appel de méthodes (SLOT).
// Ajouter dans Mafenetre.h

```
public slots:
     void convertirKmMilles();
     void convertirMillesKm();
     void desactiverMilles();
     void desactiverKm();
```

// Ajouter dans le constructeur

```
connect(lineEditKm1,SIGNAL(editingFinished()),this,SLOT(convertirKmMilles()));
connect(lineEditMilles2,SIGNAL(editingFinished()),this,SLOT(convertirMillesKm()
));
    connect(radioKmM,SIGNAL(clicked()),this,SLOT(desactiverMilles()));
    connect(radioMKM,SIGNAL(clicked()),this,SLOT(desactiverKm()));
```

Compte - Rendu :
Expliquez ces 4 lignes de connect, en utilisant la fiche signal et slot. :
http://doc.qt.io/qt-4.8/signalsandslots.html

//Code de la fonction

```
void ConvKM::convertirKmMilles()
{
    QString txt =lineEditKm1->text();
    double val = txt.toDouble();
    val*= 0.621371;
    lineEditMilles1->setText(QString::number(val));

}

void ConvKM::convertirMillesKm()
{

//compléter le code en vous inspirant de convertirKmMilles()

}
void ConvKM::desactiverMilles()
{
```

```
    lineEditKm1->setText("");
    lineEditMilles1->setText("");
    lineEditKm1->setEnabled(true);
    lineEditMilles2->setEnabled(false);
}

void ConvKM::desactiverKm()
{
//compléter le code en vous inspirant de desactiverMilles()
}
```

Lancer le programme et tester le convertisseur

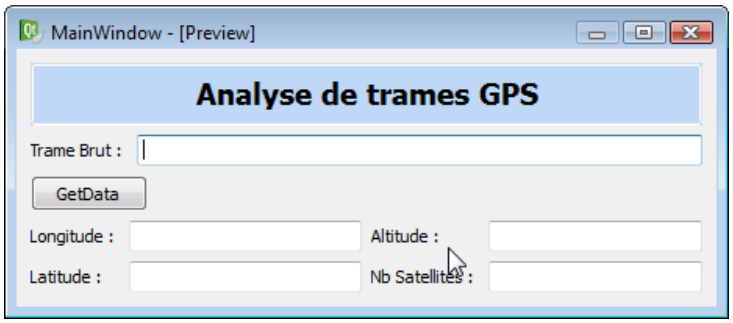
Écrire le code du destructeur de la classe ConvKm

Compte - Rendu :
Copiez le code de ConvKm.cpp
Mettre une copie d'écran du résultat

5 – Exercice complet : Décoder trame GPS : (14 pts)

En utilisant, les compétences acquises dans la première partie du TP, réaliser un code qui traite une trame GPS. Le fichier trame.txt contient des exemples de trames GPS.
Dans l'étape 1 (5 pts), La trame GPS est copiée dans le QlineEdit.
Dans L'étape 2 (4 pts), Lors de l'appui sur le bouton, la première ligne du fichier est lue.
Dans L'étape 3 (3 pts), Choix du fichier grâce à QfileDialog.
Dans L'étape 2 (2 pts), Bouton **Sauvegarde**, permet d'ajouter des lignes dans un fichier csv.

Étape 1 : IHM + décode trame GPS (5 pts)



Vous utiliserez la méthode split(...) de QString afin de découper la trame en plusieurs sous-chaines de caractères. Vous obtiendrez un QStrinList qui est une sorte de tableau (liste chainée en réalité) de QString.

Dans la classe QStringList, vous exploiterez la méthode size(..) permettant de connaître le nombre d'éléments dans la liste et la surcharge de tableau […] pour accéder plus facilement aux éléments de la liste.

A savoir : QStrinList hérite de QList qui est une classe permettant d'obtenir des listes de tout ce que l'on veut (QString, QFile, etc.....)

Étape 2 : Extrait d'une trame GPS (4 pts)

Dans L'étape 2, Lors de l'appui sur le bouton, la première ligne du fichier est lue.

- Lire le contenu du fichier tramegps1.txt (une seule ligne)
- Écrire cette trame dans un lineEdit

Vous utiliserez les classe suivantes :

Qfile	http://doc.qt.io/qt-4.8/qfile.html
QtextStream	http://doc.qt.io/qt-4.8/qtextstream.html

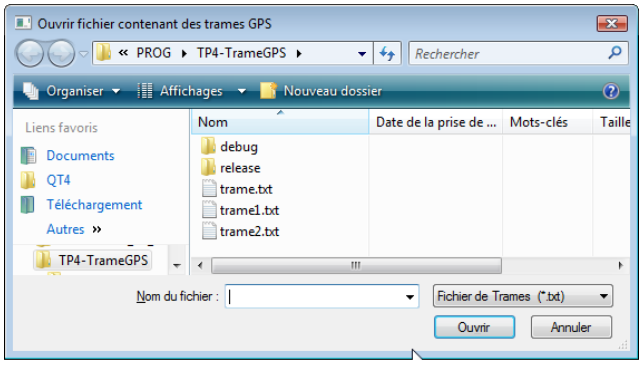
– Fin de la partie notée sur 15 /20 –

Compte - Rendu :

Copier le code de l'application
Mettre une copie d'écran du résultat

Etape 3 – QFileDialog (3 pts)

Vous ajouterez du code bouton "GetData"
Ce bouton devra ouvrir une fenêtre de *dialogue d'ouverture de fichier* . Vous utiliserez pour cela la classe QfileDialog.
Ainsi l'utilisateur pourra ouvrir le fichier qu'il souhaite.



Etape 4 – Sauvegarde des données (2 pts)

Vous ajouterez un bouton "Sauvegarde" en bas à droite de l'IHM. Ce bouton enregistrera directement (sans QFileDialog) dans un fichier texte : DataTrame.csv, les 4 infos extraites de la trame.

Le format d'enregistrement dans le fichier sera :
longitude;latitude;altitude;NbSatellite; suivi d'un retour à la ligne.

Les ; (points-virgule) serviront à séparer les données lors de l'importation du fichier dans un tableur.

ATTENTION ! Lors de plusieurs click successifs sur le bouton de sauvegarde, les données enregistrées devront se suivre et ne pas écraser les anciennes.

Compte - Rendu :

Copier le code final de l'application

Mettre une copie d'écran du résultat

- **Enregistrer le compte rendu sous CRQtTP1CPPvotreNom.pdf, rendre ce compte-rendu via Pronote avant le mardi 2 Novembre**