



Initiation au logiciel de développement d'application Rapide (RAD) en C++

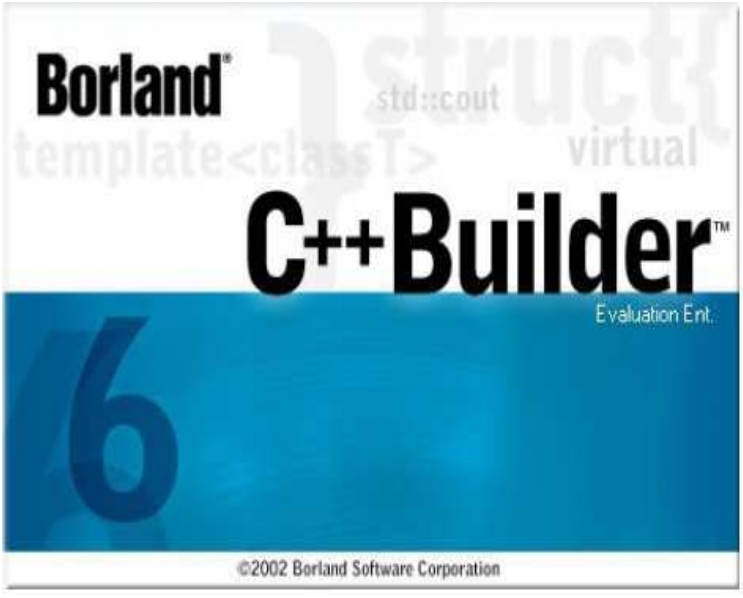




TABLE DES MATIERES

1) Présentation de Borland C++ builder page N°3

1.1) Les bibliothèques de Classes page N°4

1.2) Objets composants et contrôles page N°5

1.3) Propriétés, méthodes et événements page N°5

2) Notre premier programme sous BORLAND C++ page N°6

3) TD N°1 conversion coordonnées cartésienne_polaire page N°12

4) **TD N°2 Mise en place d'un applicatif** page N°14
GUI sous l'environnement QT, LINUX

5) TDN°3 Dialogue avec le simulateur page N°39

6) TD N°4 : Développement D'un simulateur pour page N°54

7) TD N°5 : Développement d'une IHM pour la gestion du port série page N°67

7) Les composants de la bibliothèque VCL page N°79

8) Les composant VCL du package IOCOMP page N°88

9) Quelques fonctions utiles BC++ page N°91

Initiation au logiciel Borland C++ Builder

1) Présentation de Borland C++ builder

Borland C++Builder est un environnement de programmation visuelle orienté objet permettant le développement d'applications 32 bits en vue de leur déploiement sous Windows et sous Linux. En utilisant C++Builder, vous pouvez créer de puissantes applications avec un minimum de programmation. C++Builder propose un ensemble d'outils de conception pour le développement rapide d'applications (RAD), dont des experts programmateur et des modèles d'applications ou de fiches, et gère la programmation orientée objet avec deux bibliothèques étendues de classes :

• La *bibliothèque de classes VCL* comprend des objets qui encapsulent l'API Windows ainsi que d'autres techniques de programmation utiles (Windows).

• La *bibliothèque de composants Borland multiplate-forme* (CLX), qui contient des objets encapsulant la bibliothèque Qt (Windows ou Linux).

Démarrage

Vous pouvez démarrer C++Builder de plusieurs façons :

• Double-cliquer sur l'icône C++Builder (si vous avez créé un raccourci).

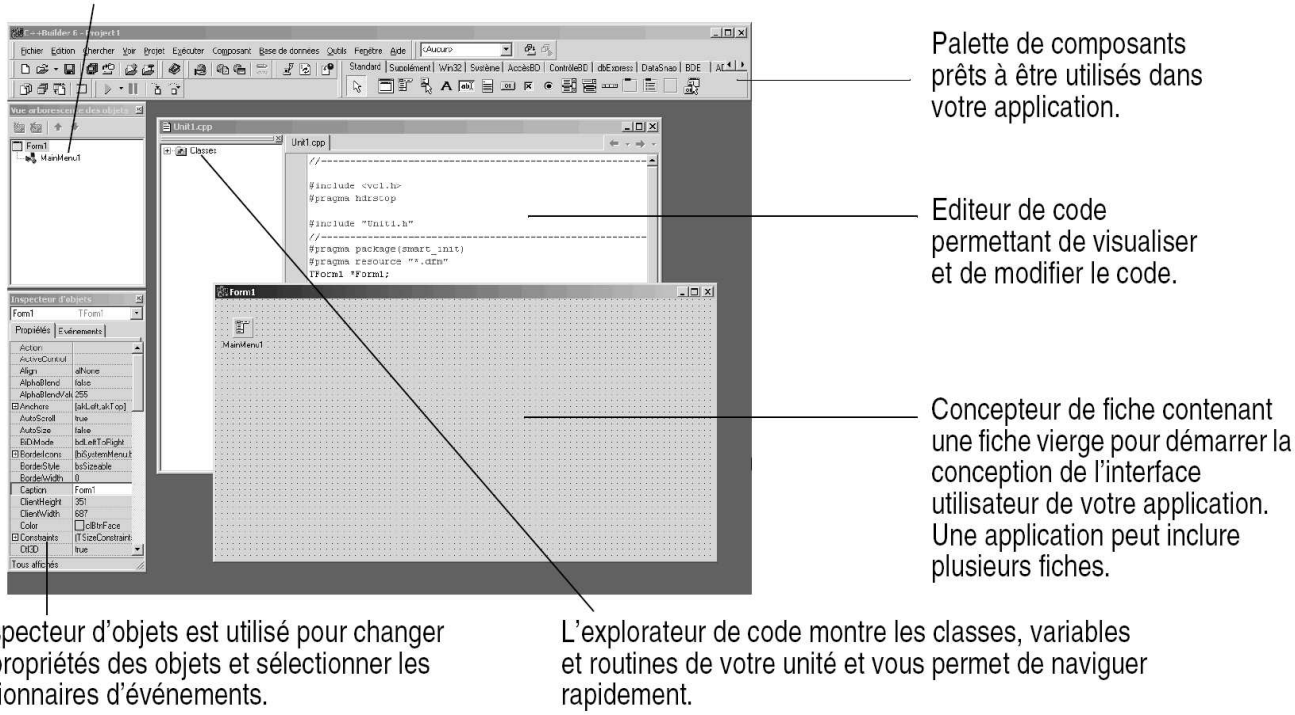
• Choisir Programmes|Borland C++Builder dans le menu Démarrer de Windows.

• Choisir Exécuter dans le menu Démarrer de Windows, puis entrer Bcb.

• Double-cliquer sur Bcb.exe dans le répertoire CBuilder6\Bin.

L'EDI (environment development interface)

Lorsque vous démarrez C++Builder pour la première fois, vous voyez apparaître quelques-uns des principaux outils proposés dans l'EDI. Dans C++Builder, l'EDI comprend des barres d'outils, des menus, la palette de composants, l'inspecteur d'objets, l'arborescence d'objets, l'éditeur de code, l'explorateur de classes, le gestionnaire de projet et bien d'autres outils. Selon l'édition de C++Builder, les caractéristiques et les composants disponibles varient.

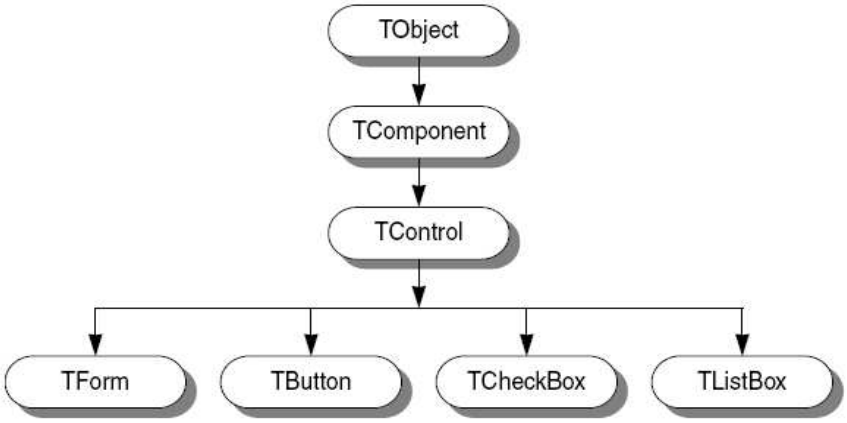


2) Les bibliothèques de classes

C++Builder comprend à la fois la bibliothèque de composants visuels (VCL) et la bibliothèque de composants Borland multiplate-forme (CLX). La VCL est réservée au développement Windows tandis que CLX permet le développement multiplate-forme Windows et Linux. Ces deux bibliothèques de classes sont différentes mais présentent de nombreuses similarités.
Tous les objets VCL et CLX dérivent de *TObject* qui est une classe abstraite dont les méthodes encapsulent des comportements essentiels comme la construction, la destruction et la gestion des messages. Quand vous écrivez vos propres classes, elles doivent dériver de *TObject* dans la bibliothèque de classes que vous prévoyez utiliser. *Les composants* sont un sous-ensemble de la VCL ou de CLX et dérivent de la classe abstraite *TComponent*. Vous pouvez placer les composants dans une fiche ou un module de données et les manipuler pendant la conception. La plupart des composants sont visuels ou non, selon qu'ils apparaissent ou pas à l'exécution.

3) Objets, composants et contrôles

La figure suivante est une vue très simplifiée de la hiérarchie des héritages qui
Illustre les relations entre objets, composants et contrôles.



4) Propriétés, méthodes et événements

VCL et CLX sont toutes deux une hiérarchie d'objets, intégrée à l'EDI. Chaque objet contient des données membres (propriétés), des fonctions qui opèrent sur les données (méthodes) et un moyen d'interagir avec les utilisateurs des classes (événements).

a) Propriétés

Les *Propriétés* sont les caractéristiques d'un objet (données membres), relatives à son comportement visible ou aux opérations qu'il effectue. Par exemple, la propriété *Visible* détermine si un objet doit être vu ou non dans l'interface d'une application. Des propriétés bien conçues simplifient l'utilisation de vos composants par d'autres programmeurs et en facilite la maintenance.

b) Méthodes

Une *méthode* est une fonction qui est un membre d'une classe. Les méthodes définissent le comportement d'un objet. Les méthodes de classe peuvent accéder à toutes les propriétés *publiques*, *protégées* et *privées* et aux données membres de la classe

c) Les événements

Un *événement* est une action ou une occurrence détectée par un programme. La plupart des applications modernes sont dites pilotées par événements, car elles sont conçues pour répondre à des événements. Vous allez donc écrire le code qui gère chacun des événements qui vous intéressent. Quelle que soit la façon dont un événement a été appelé, C++Builder recherche si du code a été écrit pour gérer cet événement. Les types d'événements qui peuvent survenir se divisent en deux grandes catégories :

- Evénements utilisateur
- Evénements système

La section suivante fournit une analyse du développement de logiciels avec C++Builder : création d'un projet, travail sur les fiches, écriture de code, compilation, débogage, déploiement des programmes.

5) Notre premier programme

Cliquez donc sur **Fichier** puis **Enregistrer Tout**. C++Builder vous présentera alors une boite de dialogue proposant d'enregistrer le fichier courant sous le nom de Unit1.cpp. Créez un dossier nommé **Test** dans le répertoire **C:\Borland** et enregistrez-y le fichier sous le nom de **Test1_main.cpp** pour bien repérer que ce fichier est le fichier principal de votre projet. Validez. C++Builder vous propose maintenant d'enregistrer un nouveau fichier sous le nom de **Project1.bpr**. Un fichier .bpr est un fichier projet. Changez ce nom et nommez-le **Test1.bpr**. Ainsi, le fichier **Test1.bpr** a bien été créé, de même que le fichier **Test1_main.cpp**.

Mais quels sont ces autres fichiers. Le fichier **Test1.cpp** est le fichier correspondant au fichier projet **Test1.bpr**. Le fichier **Test1.cpp** contient le code qui permettra à votre programme de démarrer ; nous étudierons ce code plus tard. Le fichier **Test1_main.cpp** contient le code correspondant à la fiche (=fenêtre) principale. Cette fenêtre principale se construit facilement. Toutes les informations la concernant se trouvent dans le fichier **Test1_main.dfm**. On constate que C++Builder a également créé un fichier nommé **Test1_main.h** qui sera le fichier d'en-tête contenant toutes les déclarations correspondantes à cette fiche principale.

a) Premiers pas

Revenons dans C++Builder, où nous nous trouvons face à une zone de texte qui contient du code que nous détaillerons plus tard, pour le moment, ce n'est pas très important. Avec l'explorateur, nous pouvons aller faire un petit tour dans le répertoire **C:\Borland\Test** qui contient quelques fichiers :



Qu'est-ce qu'un fichier projet ?
Lorsque vous développez (ou développerez) correctement un logiciel assez conséquent, vous avez un certain nombre de fichiers, de librairies qui doivent être tous liés ensembles. Ainsi, le fichier projet signale quels fichiers doivent être liés, quelles options doivent être passées au compilateur, ...

Ainsi, le fichier **Test1.bpr** a bien été créé, de même que le fichier **Test1_main.cpp**. Mais quels sont ces autres fichiers. Le fichier **Test1.cpp** est le fichier correspondant au fichier projet **Test1.bpr**. Le fichier **Test1.cpp** contient le code qui permettra à votre programme de démarrer ; nous étudierons ce code plus tard. Le fichier **Test1_main.cpp** contient le code correspondant à la fiche (=fenêtre) principale. Cette fenêtre principale se construit facilement.

Toutes les informations la concernant se trouvent dans le fichier **Test1_main.dfm**. On constate que C++Builder a également créé un fichier nommé
Test1_main.h qui sera le fichier d'en-tête contenant toutes les déclarations correspondantes à cette fiche principale.

Revenons dans C++Builder, où nous nous trouvons face à une zone de texte qui contient du code. La première méthode (__fastcall Form1::TForm1(TComponent* Owner)) qui vous est donné est le constructeur de l'objet feuille ou fiche form1.

b) La palette des composants

Nous allons commencer par faire un programme tout simple, qui ne nous demandera de taper qu'une unique ligne de code. Appuyez sur la touche [F12]. Une fenêtre apparait, parsemée de petits points qui forment un réseau. Ces points n'apparaitront pas dans votre programme, ils sont juste là pour vous aider à positionner correctement les différents éléments qui formeront l'interface de votre programme. Vous pouvez redimensionner votre fenêtre afin qu'elle apparaisse plus petite à l'exécution. Double-cliquez ensuite sur l'icône représentant un bouton OK dans la palette des composants.



Vous pouvez vous aider de la petite bulle d'aide qui apparait et qui doit spécifier **Button**.
La palette des composants est la barre d'outils divisée en onglets et remplie de petites icônes. Ces petites icônes représentent en général le composant, comme une zone d'édition, un menu ….



Si vous avez bien double-cliqué sur l'icône représentant un bouton, vous devez voir un bouton placé au milieu de la fenêtre. Ce bouton peut être déplacé par un simple "cliquertrainer". Vous pouvez ainsi le positionner où vous le voulez. Vous pouvez également le redimensionner, mais sachez qu'il a la taille standard par défaut.

On voit apparaitre à gauche la fenêtre de l'inspecteur d'objets

c) L'inspecteur d'objets

Chaque composant comme le bouton a des propriétés telles la taille, la hauteur, la légende (titre, caption), la police qu'il est possible de modifier grâce à l'inspecteur d'objets. Si l'inspecteur d'objets n'est pas affiché, pensez à l'afficher en cliquant successivement sur **Voir**, **Inspecteur d'objets** ou tout simplement en appuyant sur **[F11]**.



Le bouton est par défaut intitulé **Button1**. Vous pouvez modifier cela en cliquant dans l'inspecteur d'objet à la ligne **Caption** et en modifiant le texte qui s'y situe. Parallèlement, le texte sera modifié sur le bouton dans la fenêtre. Modifiez le texte en **Informations**. Nous allons également changer le nom du bouton. En effet, par défaut, il s'appelle Button1. Cela peut sembler simple, mais lorsque la fiche comporte une dizaine de boutons nommés Button1, Button2, ... Button10, il est plus difficile de s'y retrouver et de savoir qui fait quoi. Retournez dans l'inspecteur d'objet et modifiez-y la valeur de la propriété **Name** en la remplaçant par **Informations_button**. Mais à quoi peut donc servir ce nom ? Il se trouve tout simplement que même si vous voyez le bouton comme un objet graphique dans une fenêtre, ce bouton est tout simplement une variable ; retournez dans la zone d'édition et cliquez sur l'onglet **Test1_main.h** puis cliquez sur. Voilà normalement ce que vous devez y voir :

```
//---------------------------------------------

#ifndef Test1_mainH
#define Test1_mainH
//---------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------
class TForm1 : public TForm
{
__published:    // Composants gérés par l'EDI
    TButton *Informations_button;
private:        // Déclarations utilisateur
public:         // Déclarations utilisateur
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------
#endif
```

Vous remarquerez rapidement à la ligne 14 le code TButton *Informations_button. **Informations_button** est bien le nom de notre bouton et tout simplement une variable. Cela permet d'introduire la notion de RAD : lorsque vous avez placé le bouton sur la fiche, vous l'avez fait rapidement. Et vous n'avez pas eu à placer le code déclarant cette variable pointeur **Informations_button** ; cela s'est fait tout seul. C++Builder crée donc le code pour vous afin que vous ne perdiez pas de temps à coder des lignes qui relèvent de la routine et non véritablement de la conception du projet. Nous allons maintenant écrire une ligne de code afin que notre programme fasse quelque chose. Si vous le compilez et l'exécutez maintenant, vous auriez seulement un bouton sur une fenêtre et le clic sur ce bouton ne donnerait aucun résultat. (Vous pouvez essayer pour voir, appuyez sur **[F9]**. Après quelques secondes, le programme se lance ; fermez-le ensuite)
La ligne de code que nous allons écrire va permettre d'afficher une petite boite de message avec quelques informations lors d'un clic sur le bouton I**nformations_button**. Revenez à la fiche en appuyant sur **[F12]** et double-cliquez sur le bouton. La zone d'édition contient maintenant en plus le code suivant :

```
//---------------------------------------------------------------------
------
void __fastcall TForm1::Informations_buttonClick(TObject *Sender)
{

}
//---------------------------------------------------------------------
------
```

Ce code est la fonction Informations_buttonClick qui appartient à la classe TForm1. Si vous ne comprenez pas cela, ce n'est pas très grave, nous en parlerons très probablement plus tard. Ajoutez donc dans cette fonction la ligne de code suivante :

```
ShowMessage("Informations\nLogiciel développé par Toto, (c) 2002");
```

Votre fonction ressemble donc maintenant à cela :

```
//---------------------------------------------------------------------
------
void __fastcall TForm1::Informations_buttonClick(TObject *Sender)
{
    ShowMessage("Informations\nLogiciel développé par Toto, (c) 2002");
}
//---------------------------------------------------------------------
------
```

Cette simple ligne permet d'afficher un message dans une petite fenêtre. Appuyez sur **[F9]** et cliquez sur le bouton.

d) La programmation événementielle

En apprenant le C++, vous avez sûrement créé de petits programmes où des lignes de code se succédaient et s'exécutaient dans l'ordre. Ici, rien de tel. On dirait que tout marche tout seul ... C'est le principe de la programmation événementielle que vous avez mis en œuvre avec cette simple petite ligne de code : vous avez posé un bouton sur votre fiche et lorsque vous cliquez dessus, une fonction est appelée. Cette fonction est donc nommée le gestionnaire d'événement OnClick du bouton car elle réagit à l'événement "Quand on clique sur le bouton Informations_button" (=OnClick) d'où son nom automatiquement attribué par C++Builder : Informations_button**Click**. Nous allons voir cela plus clairement : appuyez sur **[F12]** pour afficher la fiche puis cliquez sur le bouton **Informations_button** pour le sélectionner. Regardez maintenant dans l'inspecteur d'objets. Vous pouvez y voir toutes les propriétés du bouton ; en haut, vous remarquerez l'onglet **Evénements** ; cliquez dessus. Vous voyez alors la liste des événements possibles pour notre bouton, avec bien évidemment, en face de l'événement OnClick notre fonction Informations_buttonClick. Notre fonction est donc bien attribuée pour gérer l'événement OnClick. Regardez la liste, vous y remarquerez l'événement OnKeyPress qui est déclenché si jamais une touche est pressée sur le bouton. Double-cliquez dans la zone correspondante, et la fonction gestionnaire d'événement OnKeyPress sera créée. Le code suivant est ajouté dans votre zone d'édition :

```
void __fastcall TForm1::Informations_buttonKeyPress(TObject *Sender,
      char &Key)
{

}
//---------------------------------------------------------------------------
------
```

Nous allons alors y placer une ligne de code :

```
ShowMessage("Vous avez appuyé sur une touche et non cliqué sur le
bouton");
```

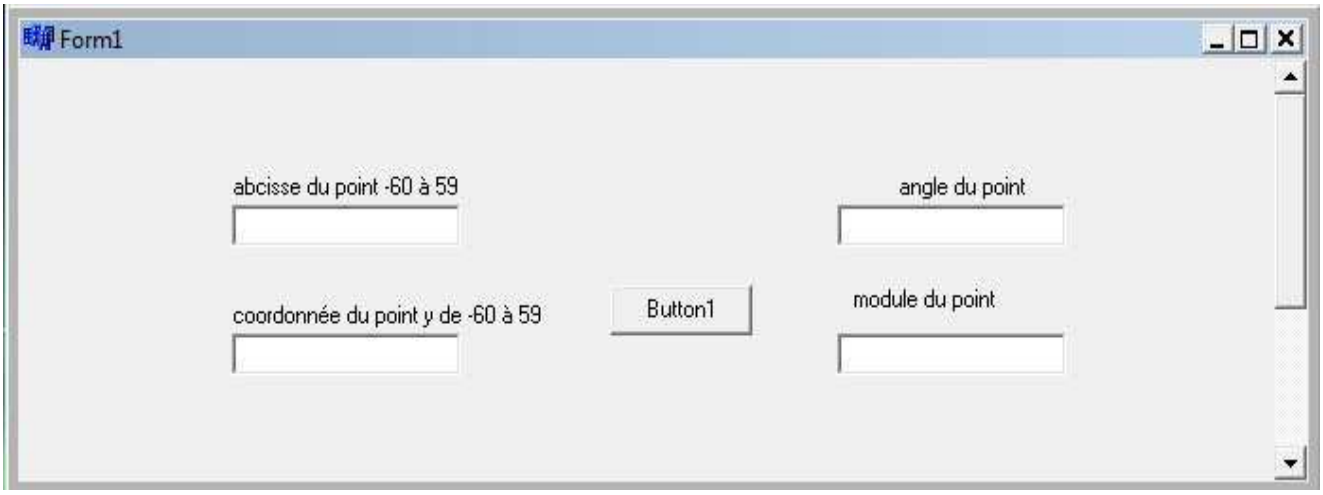
Lancez le programme et appuyez sur une touche (Remarque : évitez la touche Entrée et Espace qui sont d'une certaine manière considérés comme Windows comme des clics) : la boite de dialogue correspondante sera affichée.
Vous avez également appris qu'un programme commence toujours par une fonction main ; c'est peut-être pour cela que vous trouvez ce genre de programmation un peu bizarre. Et bien rassurez-vous, nous allons retrouver la fonction main. Cliquez donc sur le menu **Projet** puis sur **Voir le source**. Le fichier **Test1.cpp** est ouvert :

```
#include <vcl.h>
#pragma hdrstop
USERES("Test1.res");
USEFORM("Test1_main.cpp", Form1);
//---------------------------------------------------------------------
------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
         Application->Initialize();
         Application->CreateForm(__classid(TForm1), &Form1);
         Application->Run();
    }
    catch (Exception &exception)
    {
         Application->ShowException(&exception);
    }
    return 0;
}
//---------------------------------------------------------------------
------
```

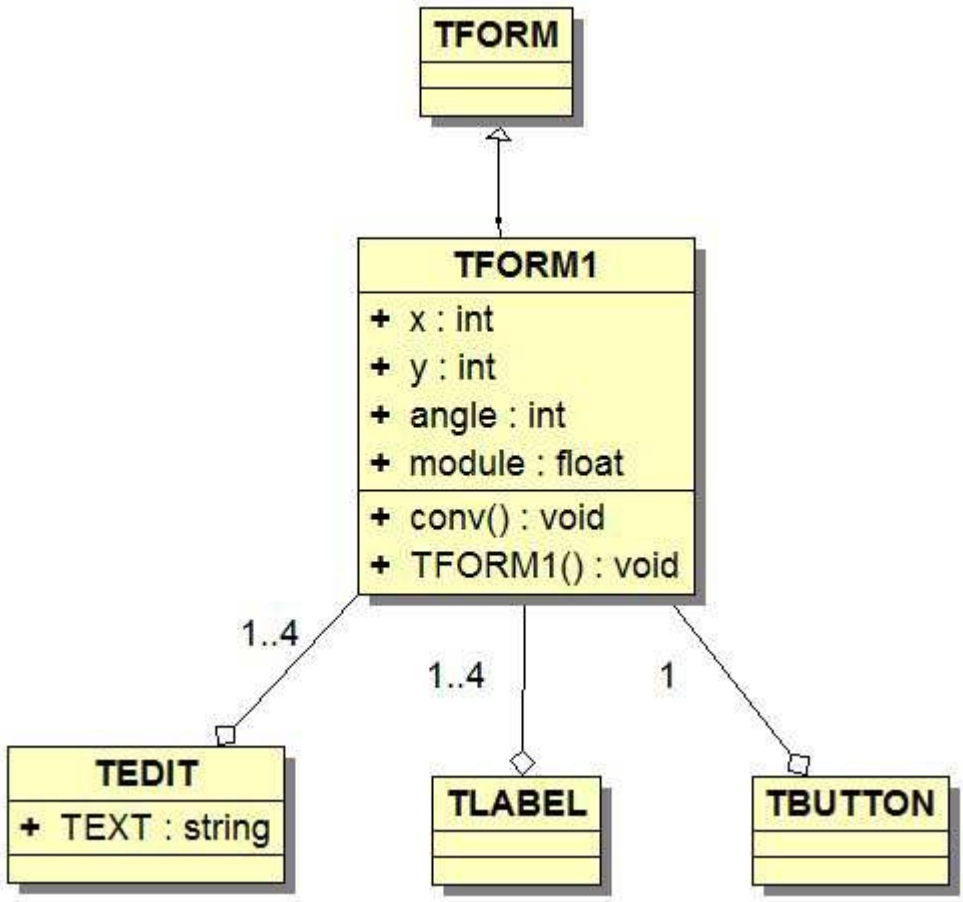
Et voilà, nous trouvons la fonction WinMain (l'équivalent pour Windows de la fonction main). Cette fonction contient le code qui permet de faire démarrer le programme. C++Builder ne présente pas ce fichier directement car on se sert rarement de la fonction WinMain, sauf dans certains cas. Nous n'allons pas détailler les blocs **try** et **catch** ; sachez seulement que si une exception (erreur grave comme la division par zéro) se produit dans le bloc try, il est possible de la gérer dans le bloc **catch**. Ici, l'erreur sera signalée (Application->ShowException (&exception);). Ce qui nous intéresse le plus pour le moment, c'est le code contenu dans le bloc **try**. Tout d'abord, Application->Initialize.Cette instruction initialise tout simplement l'application.

TD N°1

On veut créer un convertisseur de coordonnée cartésienne en coordonnées polaire. L'interface graphique sera la suivante :



On donne son diagramme de classe simplifié:



a) Quelles sont les relations qui existent entre les différentes classes.
b) Comment appelle t'on la méthode notée + TFORM1().
c) Réalisez l'interface graphique sous Borland C++, en utilisant dans la boite à composant :
   - 4 objets boîte d'édition (de classe TEDIT)
   - 4 objets LABEL (de classe TLABEL)
   - 1 objet bouton (de classe TBUTTON)
d) En analysant le code généré automatiquement par borland C++, comment peut on affirmer que les relations entre les classes (question a) sont bien celles obtenues dans le code
e) On associe à l'événement onclick() du bouton, la méthode conv(). Créez cette méthode conv (on pourra utilisez les fonctions StrToFloat et FloatToStr)
f) Testez votre programme.

TDN°2 Mise en place d'un applicatif GUI sous l'environnement QT,LINUX

1) Mon premier programme sous QT

Un programme sous QT possède toujours le code suivant :

```
#include <qapplication.h>
#include <qpushbutton.h>

int main( int argc, char **argv )
{
QApplication a( argc, argv );
return a.exec();
}
```

Sous REDHAT9 tapez le programme ci-dessus dans le répertoire /usr/tmp/TDN2

On reconnait la traditionnelle fonction main() qui est le point d'entrée du programme. La manière de procéder avec Qt, comme nous allons l'étudier, consiste à créer une instance de la classe QApplication. Un dialogue ou une fenêtre principale est ensuite instancié puis affiché avec show(). Enfin, la boucle d'évènement est installée par le lancement réel de l'application qui s'effectue par exec().

✍ Sous REDHAT9 tapez le programme ci-dessus, conclure.

Aucune fenêtre ne s'ouvre

Procédure de compilation

Dans une console placez-vous dans le répertoire /usr/tmp/TDN2 . Lancez ensuite

qmake -project && qmake && make

Lancez votre executable ./<nom du réperoire>

Pour afficher une fenêtre

Tapez le programme suivant

```
#include <qapplication.h>
#include <qpushbutton.h>

int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
fenetre.setFixedSize(640, 480);
fenetre.show();
return a.exec();
}
```

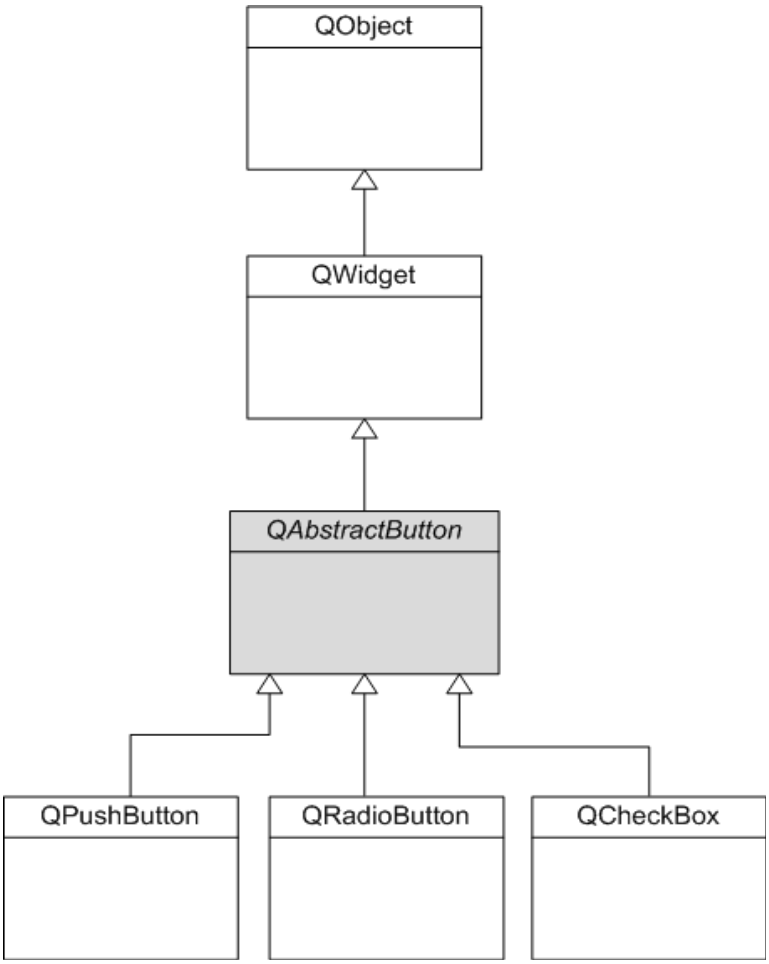
✍Sous REDHAT9 tapez le programme ci-dessus, quel est le rôle de la méthode setFixedSize

La méthode permet de fixer la dimension de l'objet fenetre

Par la suite nous allons placer différents objets graphiques sur la fenêtre, ces objets sont appelés **des widgets** .

2) Les widgets

Avec Qt, tout objet graphique de la fenêtre est appelé un **widget**. La fenêtre elle-même est considérée comme un widget.
Dans le code, les widgets sont des classes qui héritent toujours de QWidget (directement ou indirectement). C'est donc une classe de base très importante, le bouton de commande, est peut-être le widget le plus généralement utilisé dans toute interface utilisateur graphique.
Dans le code, les widgets sont des classes qui héritent toujours de QWidget (directement ou indirectement).



✍ La classe QPushButton a-t-elle un héritage direct avec la classe QWidget

Non car c'est un héritage direct avec QAbstractButton

✍ Dans le programme précédent, déclarez un pointeur bouton du type QPushButton qui aura la forme suivante et dont l'objet parent sera la fenêtre.




```
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
```

✍ Utilisez la méthode show pour afficher le bouton

```
bouton->show();
```

Cliquez sur le bouton, conclure

Aucune action car le gestionnaire d'événement n'est pas géré

3) Intéractivité entre les objets : Les signaux et les slot

Lorsqu'un objet veut signaler que quelque chose s'est passé le concernant, un bouton qui vient d'être cliqué par exemple, il émet **un signal**. Un signal est également émis par un objet lorsque son état a changé. L'objet émetteur ne sait pas comment ce signal sera interprété (il ne sera peut-être pas pris en compte). L'action à mener, le code à exécuter est contenu **dans un slot**. Les slots peuvent être employés pour recevoir des signaux, mais ce sont également des fonctions normales membres de classes. Comme nous le verrons ultérieurement, il nous suffira de connecter (associer) le signal au slot. Le programme exécutera alors le code de la méthode définie comme slot lorsque le signal d'un objet sera émis.
La fonction connect attend deux paires de paramètres. Tout d'abord l'objet émetteur du signal ainsi que le signal à surveiller. Ensuite, l'objet récepteur ainsi que le slot qui doit être déclenché.

bool QObject::connect (const QObject * *sender*, const char * *signal*, const QObject * *receiver*, const char * *method*, Qt::ConnectionType *type* = Qt::AutoConnection) [static]

```
QObject::connect(scrollBar, SIGNAL(valueChanged(int)),label,  SLOT(setNum(int)));
```

Quels sont les widgets concernés par la méthode ci-dessus
scrollBar widget émetteur, label widget récepteur
Décrire le rôle de la méthode statique connect
L'objet scrollBar envoie un signal à chaque valeur modifiée. Ce signal est intercepté par le slot et affiché par un label

On souhaite quitter l'application lorsque j'appuie sur le bouton placé précédemment.
Le signal émis sera lié à l'événement click et donc la méthode clicked(), le code à exécuter **dans un slot sera la méthode** quit()

✍ Complétez la méthode statique suivante

```
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
```

✍ Testez votre nouveau programme sous REDHAT9

✍ Dans le programme précédent, déclarez un pointeur chiffre du type QLineEdit qui aura la forme suivante et dont l'objet parent sera la fenêtre.

```
QLineEdit *vitesse=new QLineEdit("", &fenetre);
vitesse->setText("123");
```



✍ Testez votre nouveau programme sous REDHAT9
Conclure ?

Les deux objets sont placés à la même position et un des deux est non visible

4) Le problème du placement des widgets

Dans le programme précédent, déclarez un pointeur checkbox du type QCheckBox qui aura la forme suivante et dont l'objet parent sera la fenêtre.



Les 2 méthodes suivantes permettent de déplacer les objets sur la fenêtre

```
vitesse->move(100, 100);
checkbox->setGeometry(200,200,200,30);
```

✍ Testez ces 2 méthodes et décrire le rôle des méthodes move et setGeometry

La méthode move déplace de façon absolue l'objet vitesse à la position x= 100 et y=100 de la fenêtre

La méthode setGeometry déplace de façon absolue l'objet checkbox à la position x= 200 et y=200 de la fenêtre
La longueur de l'objet checkbox est de 200, sa hauteur est de 30

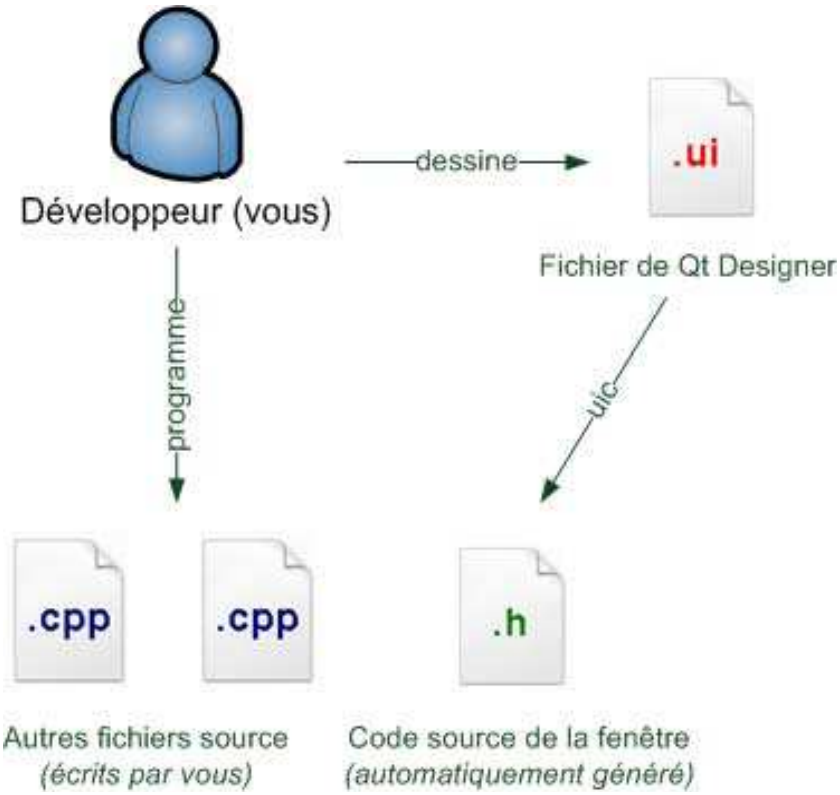
✍ Déplacez l'objet vitesse et modifiez votre programme afin d'Utilisez la méthode setText et affichez la chaine de caractère "321".

```
vitesse->move(100, 100);
vitesse->setText("123");
```

Le placement absolu des objets reste très compliqué. Une autre méthode est d'utiliser le placement relatif **avec les layouts** (ce sont des conteneurs de widgets). Cette nouvelle méthode rajoute beaucoup de code et reste assez fastidieuse.

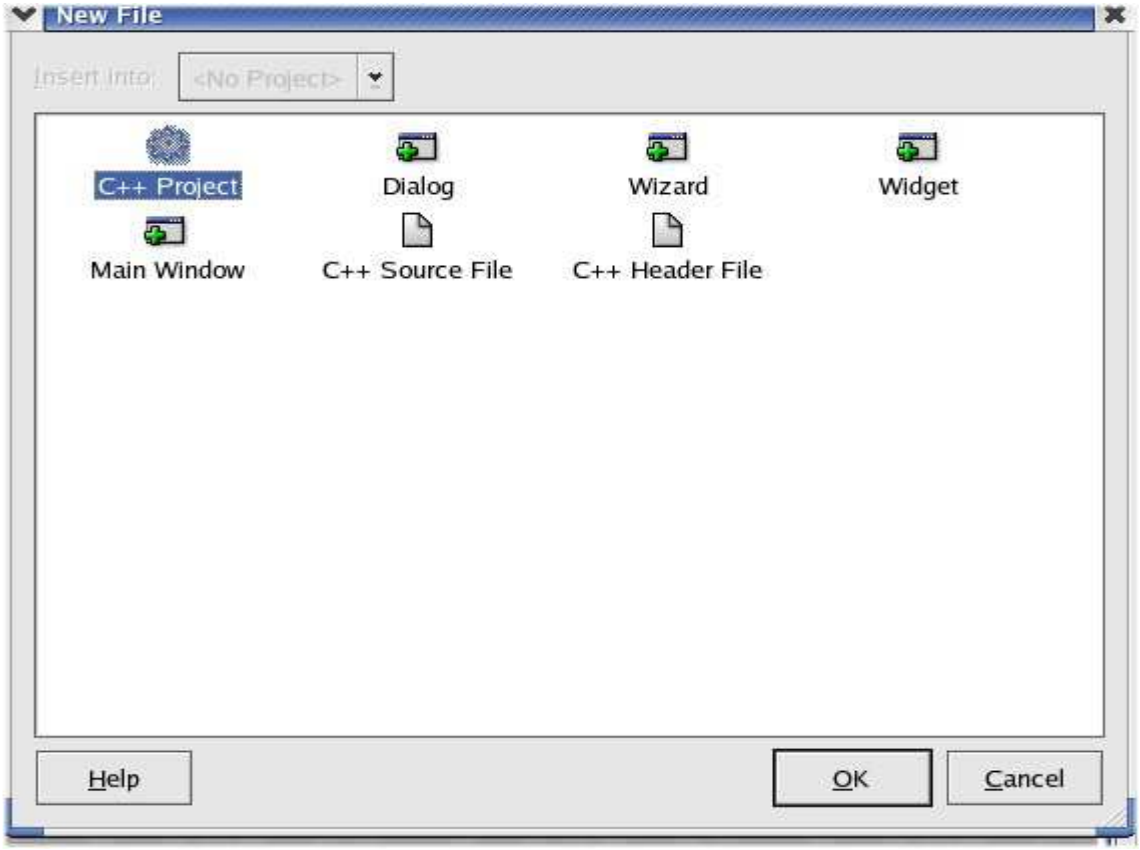
5) Mise en place d'un applicatif GUI sous QTdesigner

Qt Designer est le concepteur d'interface de Qt. Il s'agit d'un programme livré avec Qt qui permet de **dessiner vos fenêtres visuellement**. Mais plus encore, Qt Designer vous permet aussi de modifier les propriétés des widgets, d'utiliser des layouts, et d'effectuer la connexion entre signaux et slots.
Qt Designer ne fait que produire un fichier .ui. C'est le petit programme *uic* qui se charge de transformer le .ui en code source C++.

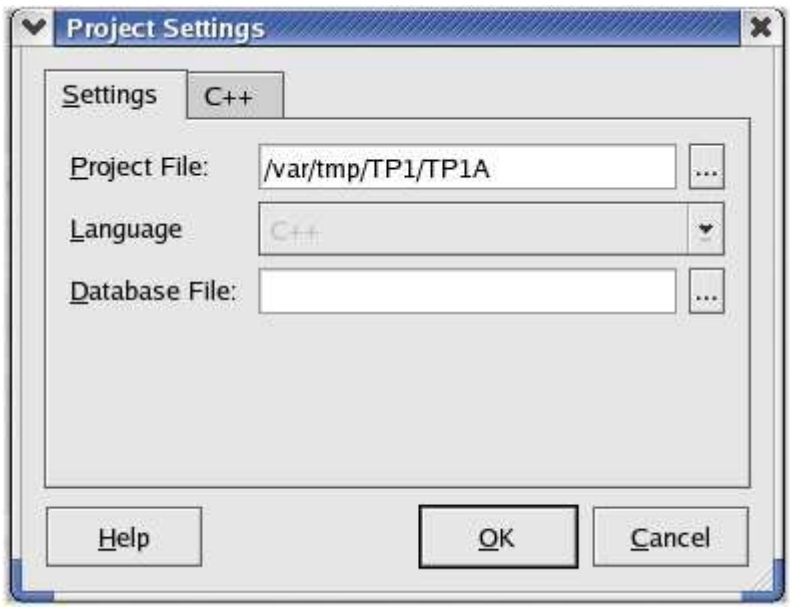


✍ Lancez le programme QTdesigner cliquez sur le chapeau →programmation →outils de programmation supplémentaire→qtdesigner

La fenêtre suivante apparait :

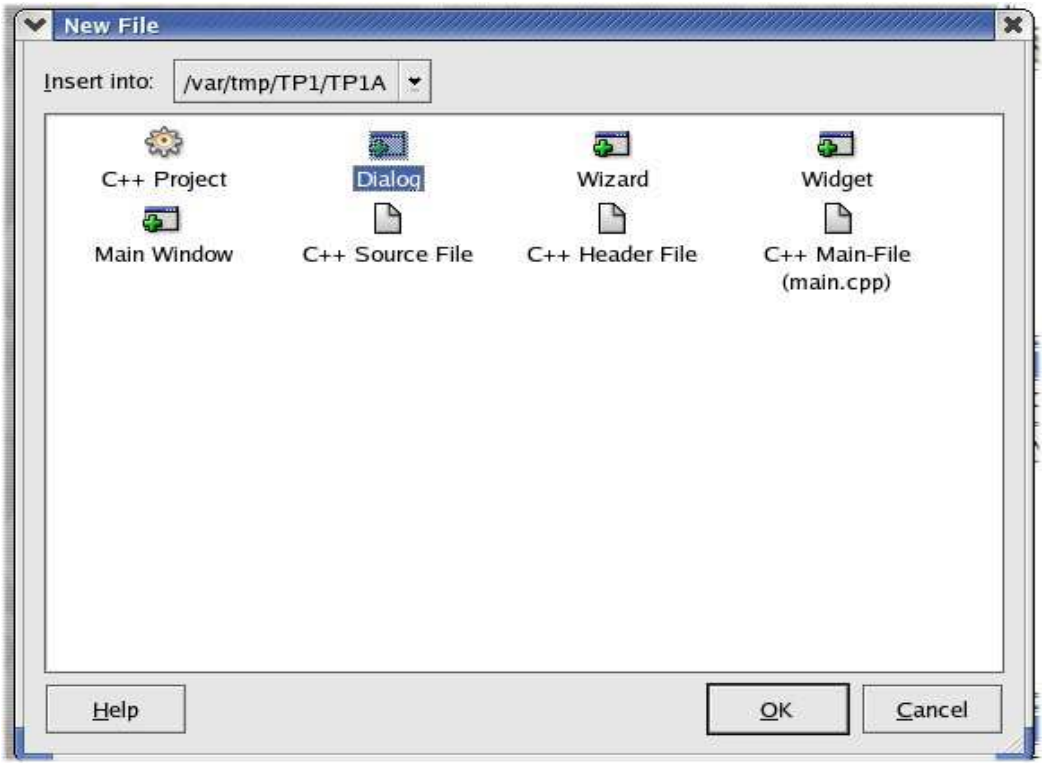


- Choisir un projet du type C++ Project

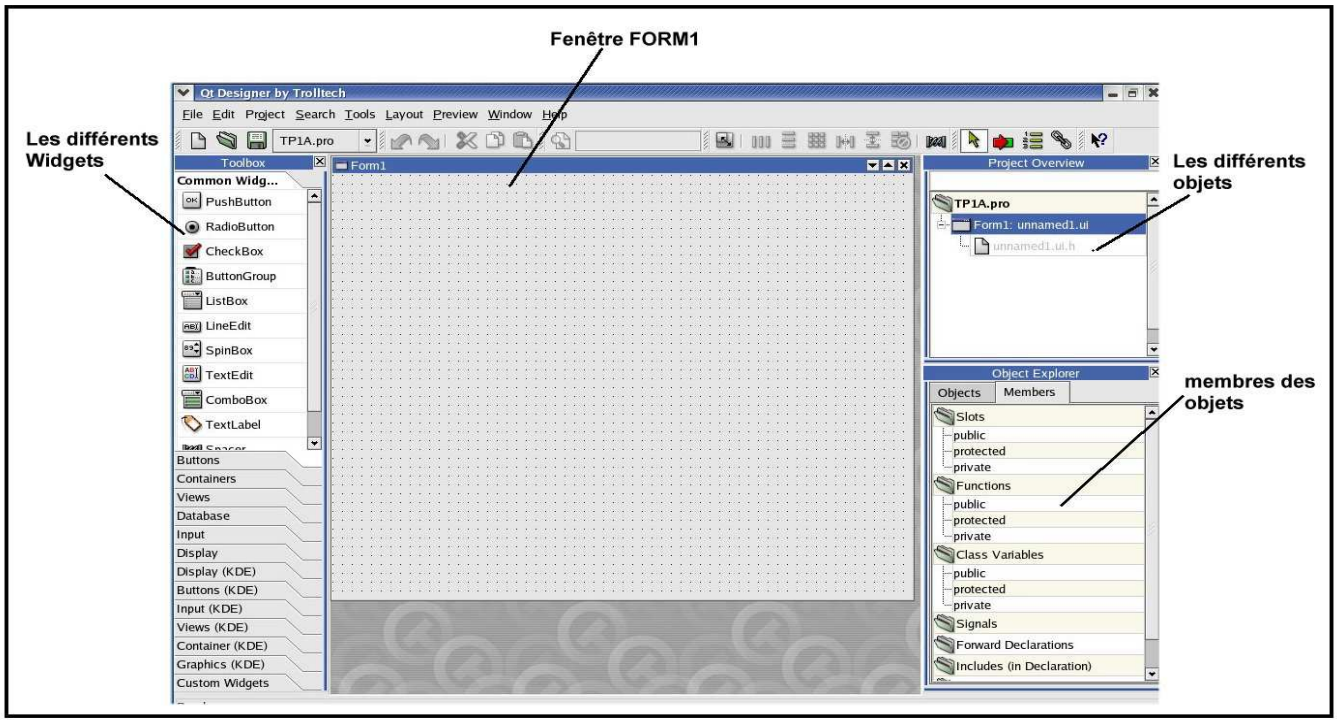


- Indiquez le nom et le chemin de votre projet /usr/tmp/TD2/TD2A

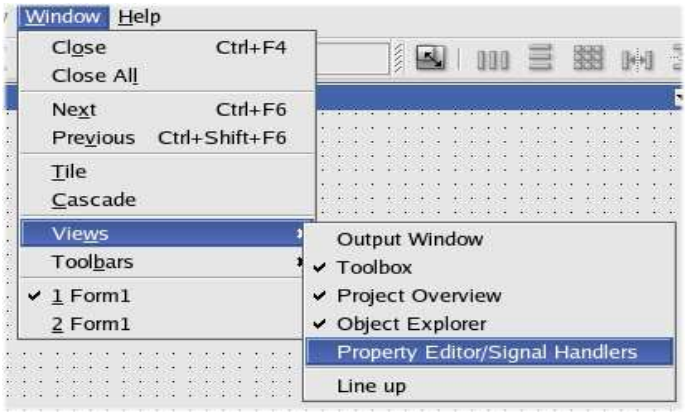
Puis le menu faire file new

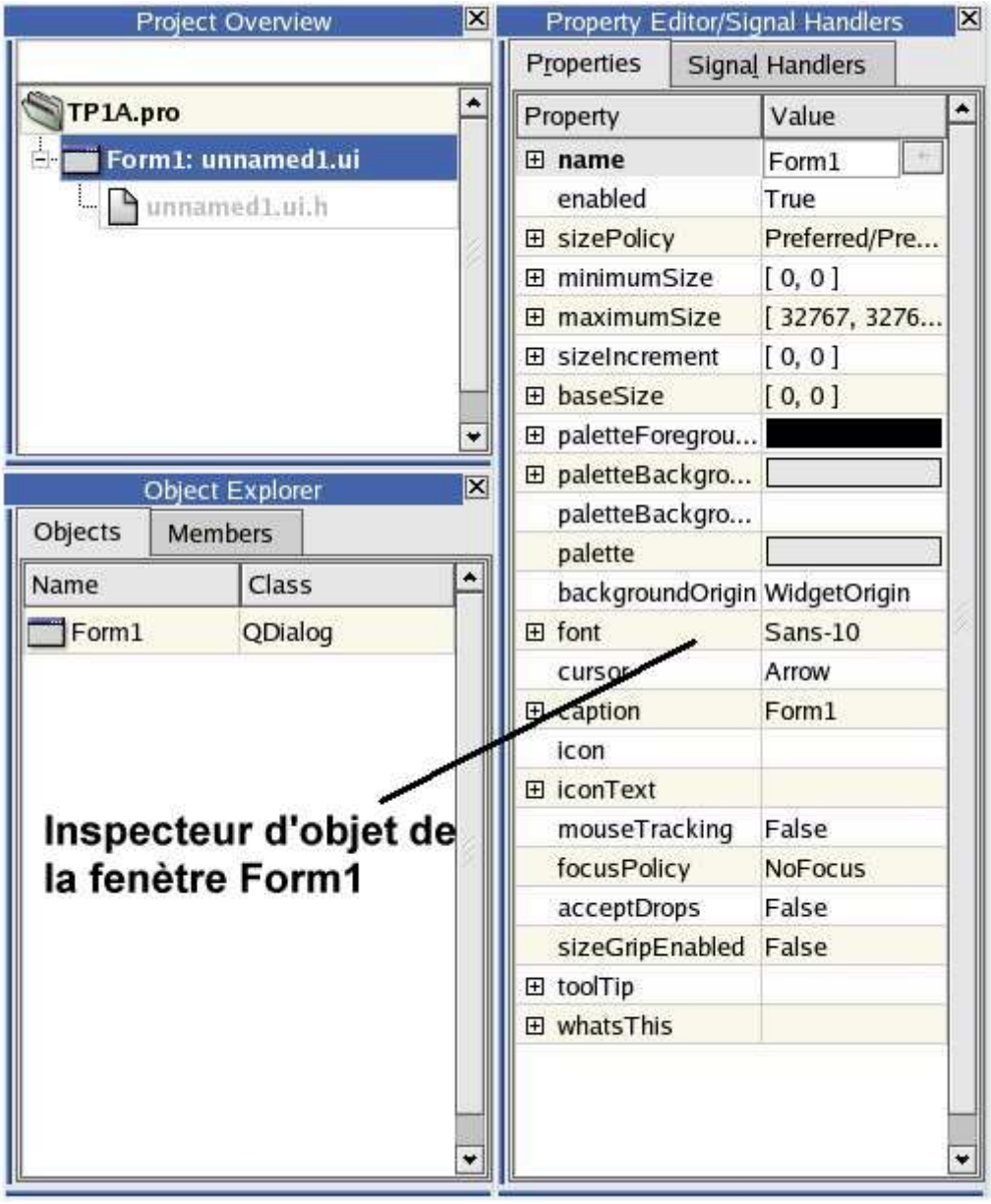


Et choisir une application du type Dialog

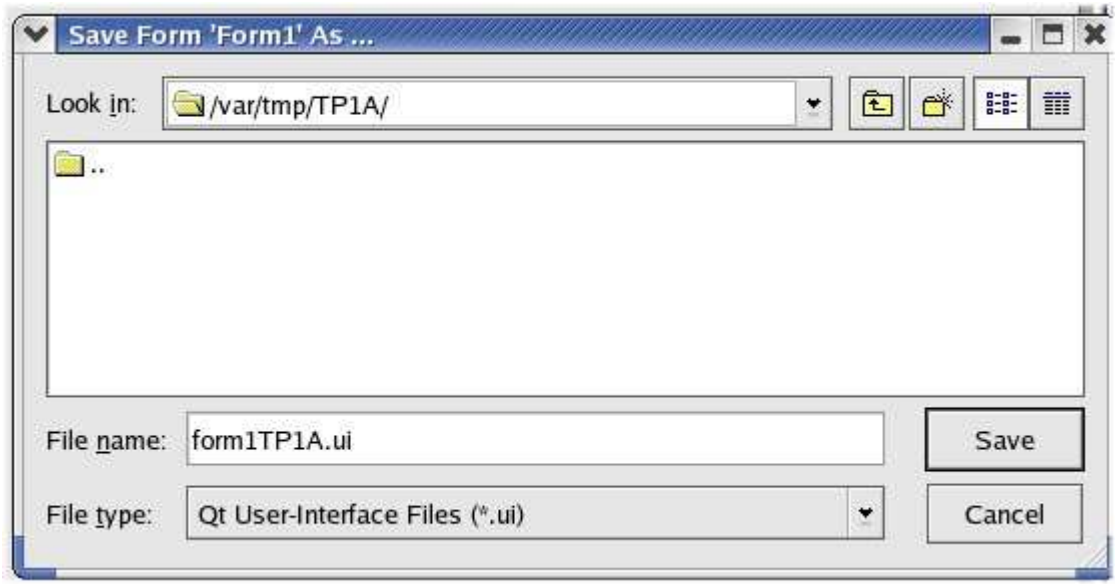


Pour ouvrir l'inspecteur d'objet procédez comme ci-dessous :

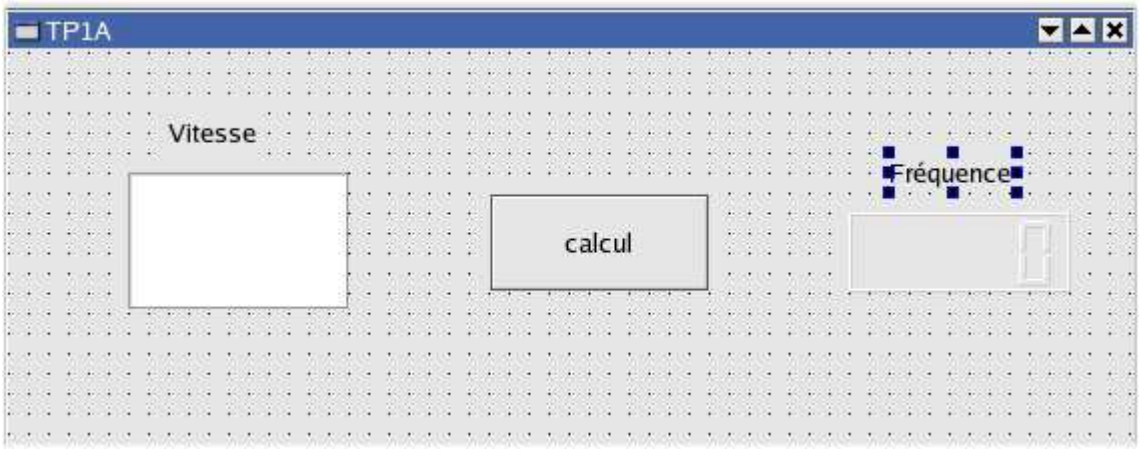


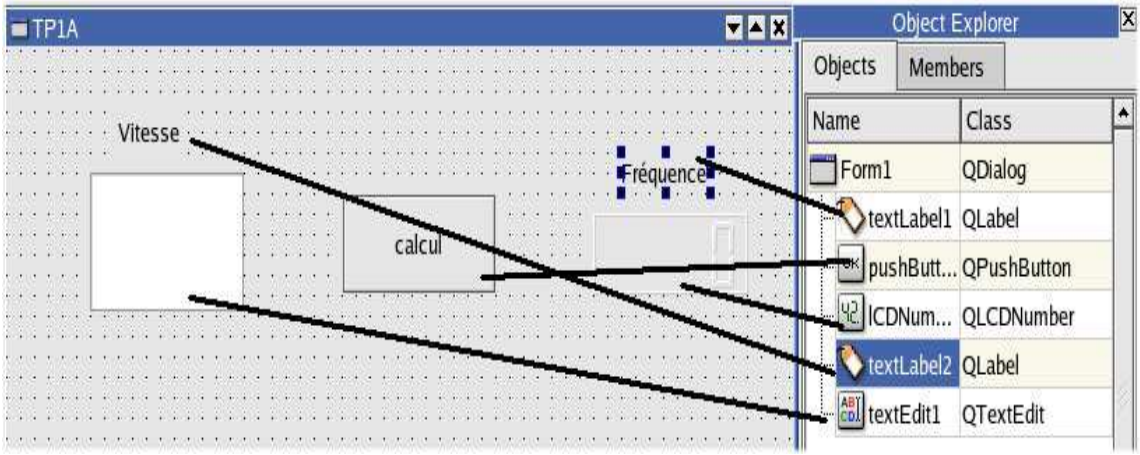


Allez dans file→save as et sauvegardez votre fichier ui sous form1TP1A.ui

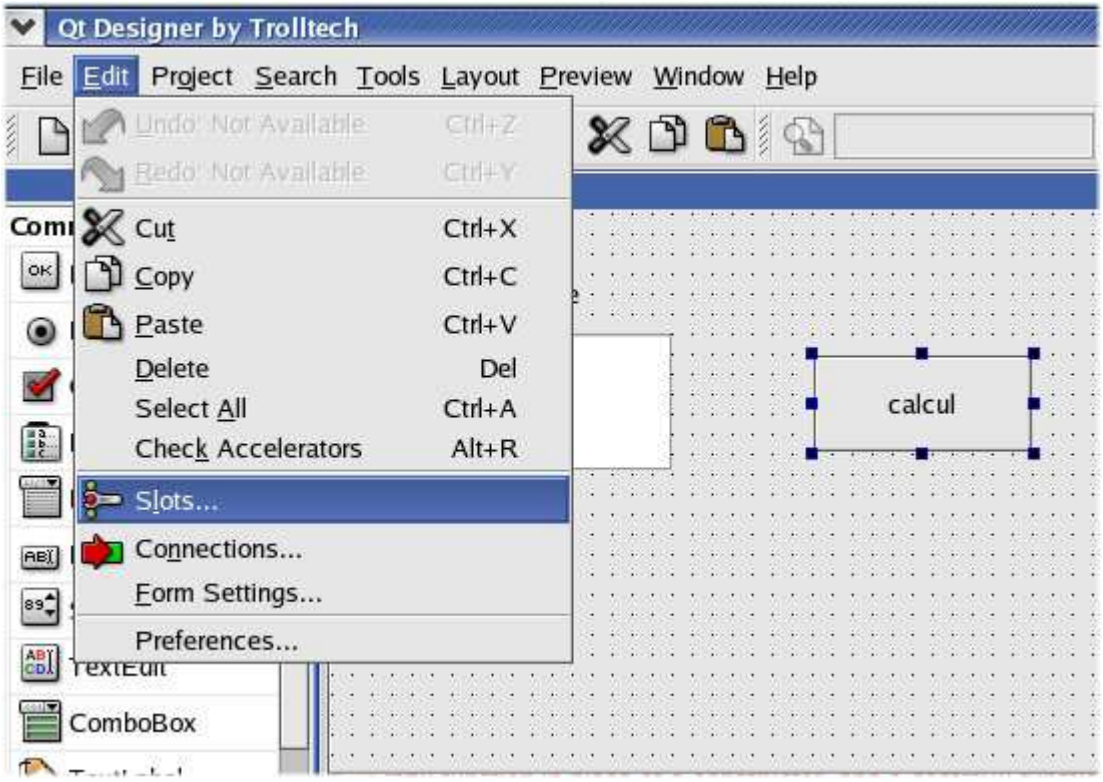


Réalisez l'interface graphique suivante :

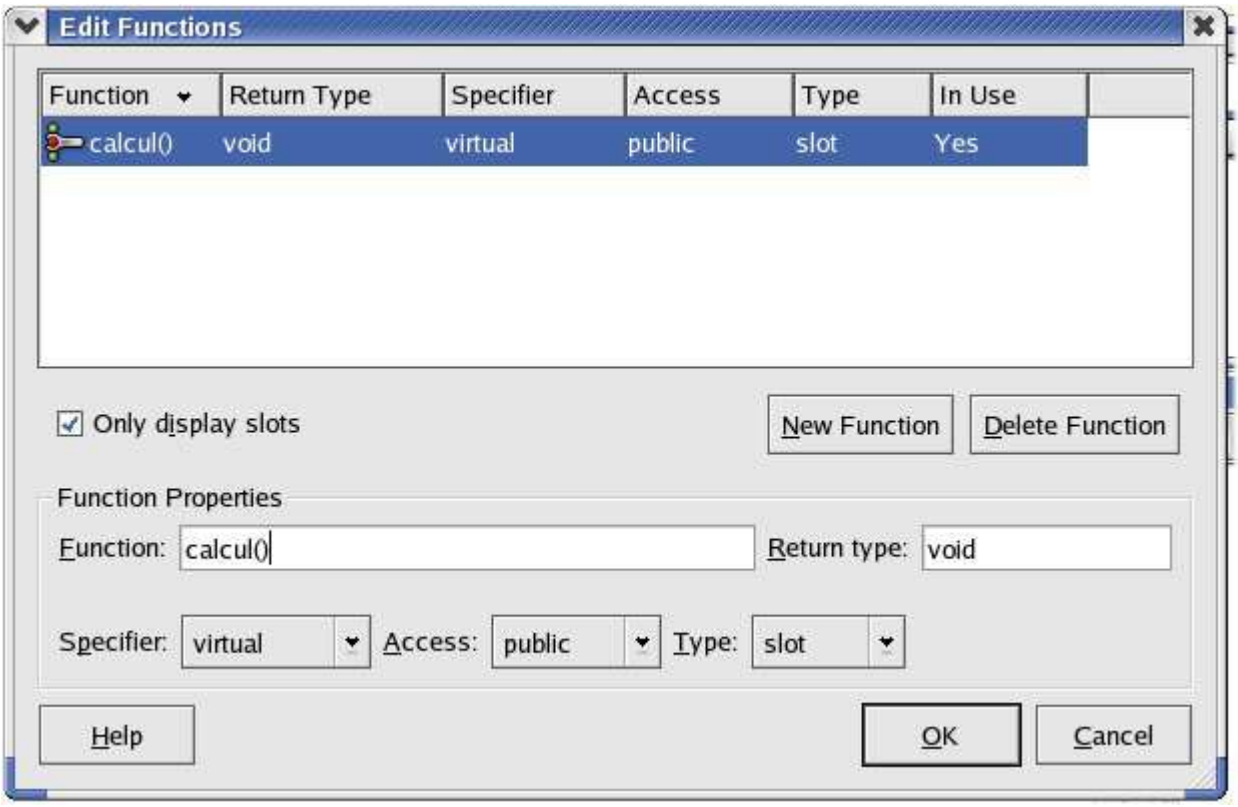




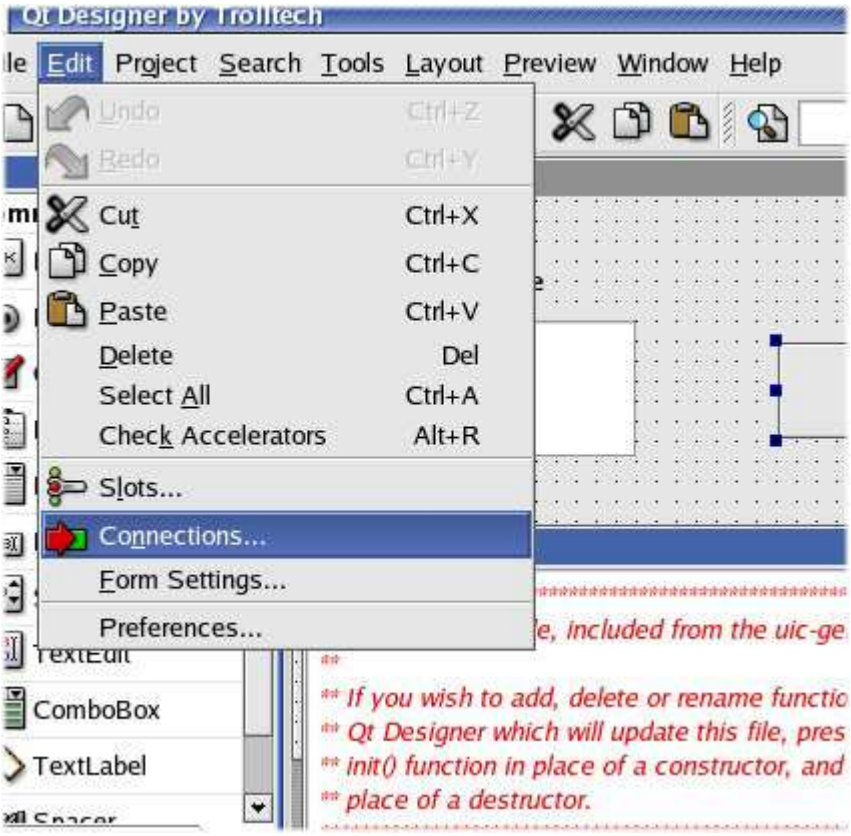
Pour créer un slot



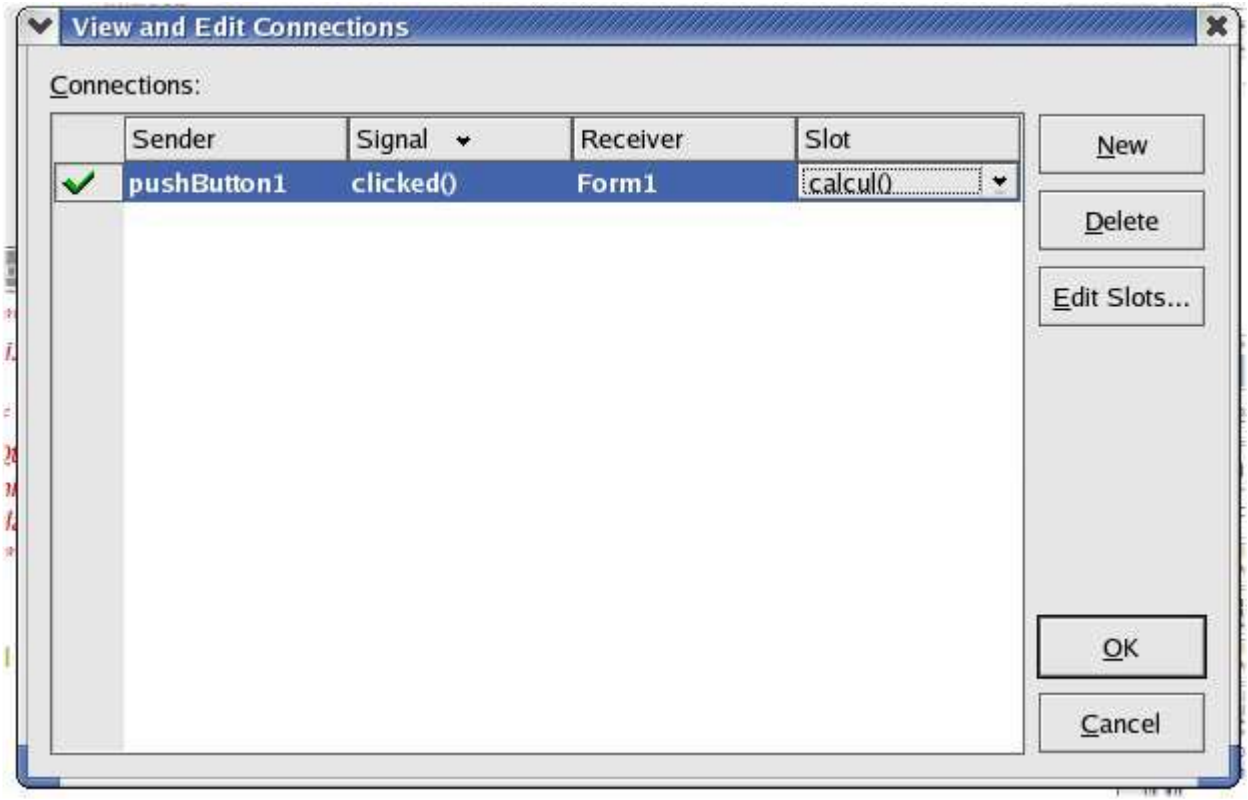
Définir la fonction calcul pour le slot



Créez la connexion entre le slot et le signal



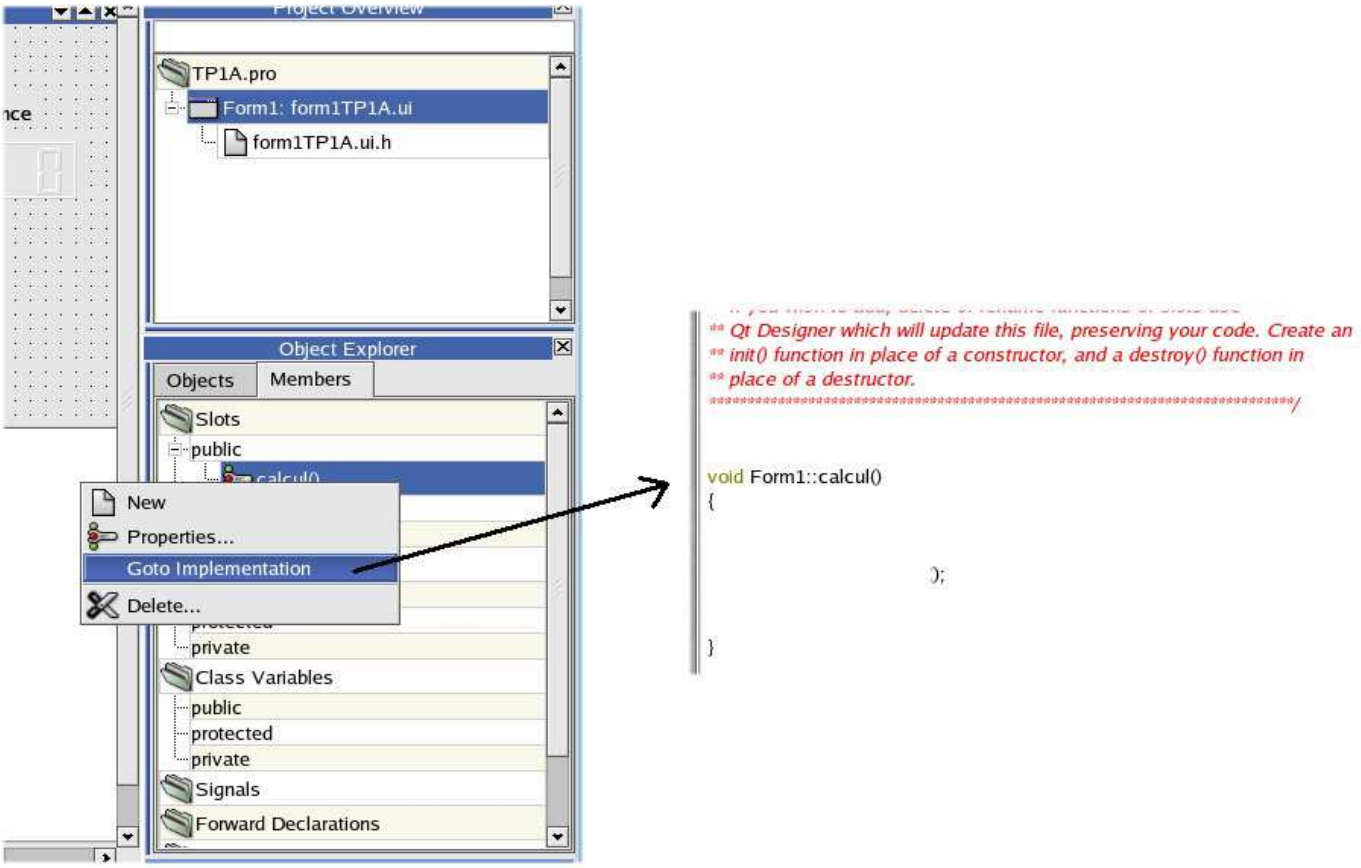
Définir le signal avec son slot



Complétez la ligne de code générée par QTdesigner

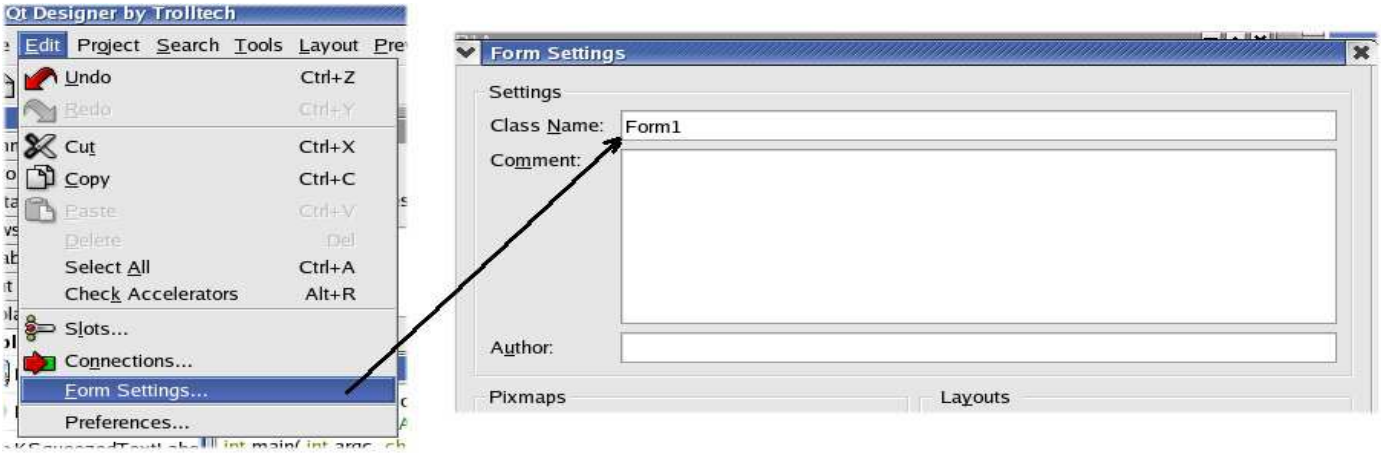
```
QObject::connect(                                                    );
```

Implantez le code de la méthode calcul, cliquez droit sur la case de la méthode calcul



Complétez le code de la fonction calcul.

Pour vérifier le nom de la classe



Afin de pouvoir compiler et exécuter le programme, nous avons besoin d'un fichier main.cpp que voici:

```
#include <qapplication.h>
#include "form1TP1A.h"
int main( int argc, char **argv )
{
QApplication a( argc, argv );
Form1 *fenetre = new Form1(0);
a.setMainWidget(fenetre);
fenetre->show();
return a.exec();
}
```

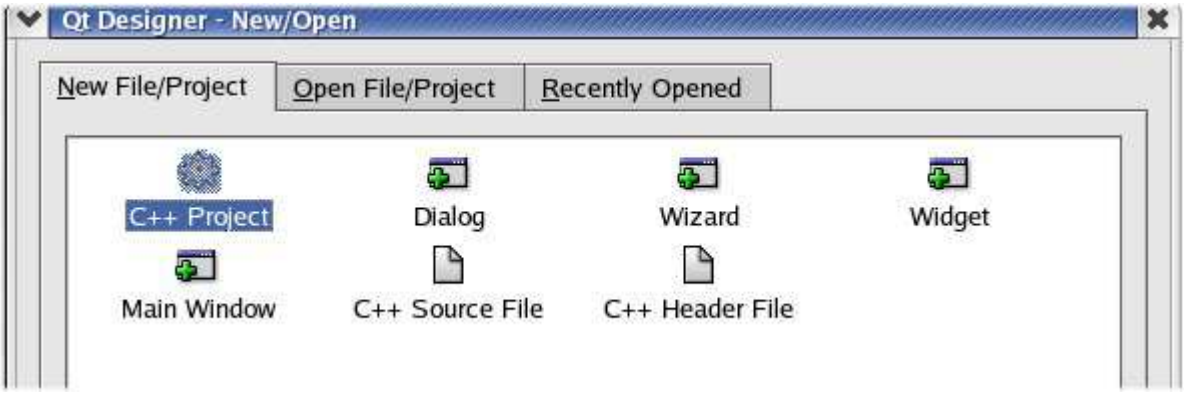
✍ Réalisez sous gedit ce programme et enregistrez le sous main.cpp

Pensez à rajouter ce fichier main.cpp au projet grâce au menu "Project/Add Files…".
Lancez dans une console, ouverte dans le répertoire du projet : qmake -project && qmake && make afin de générer un fichier Makefile puis de le compiler.

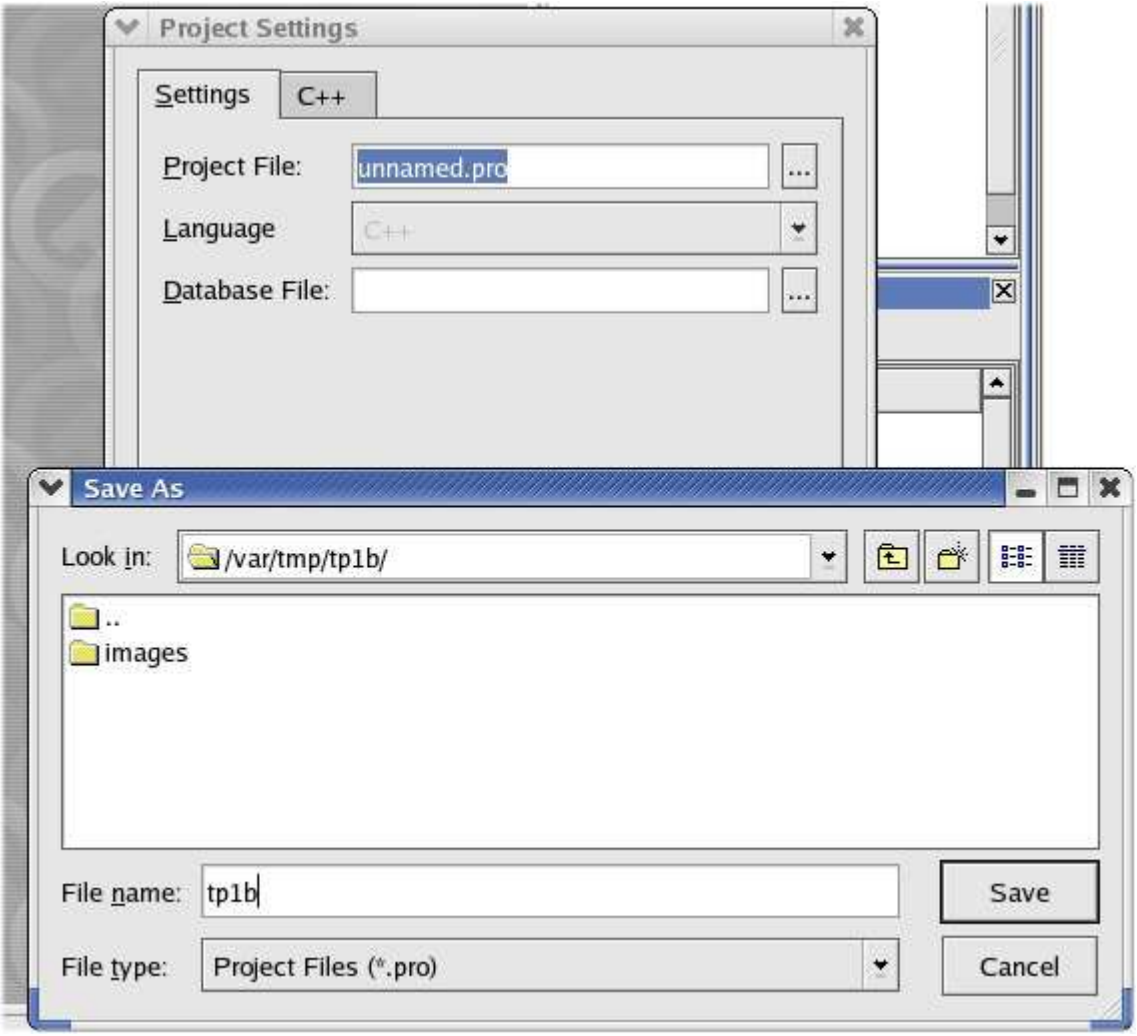
✍ Testez votre programme

Réalisation d'un menu

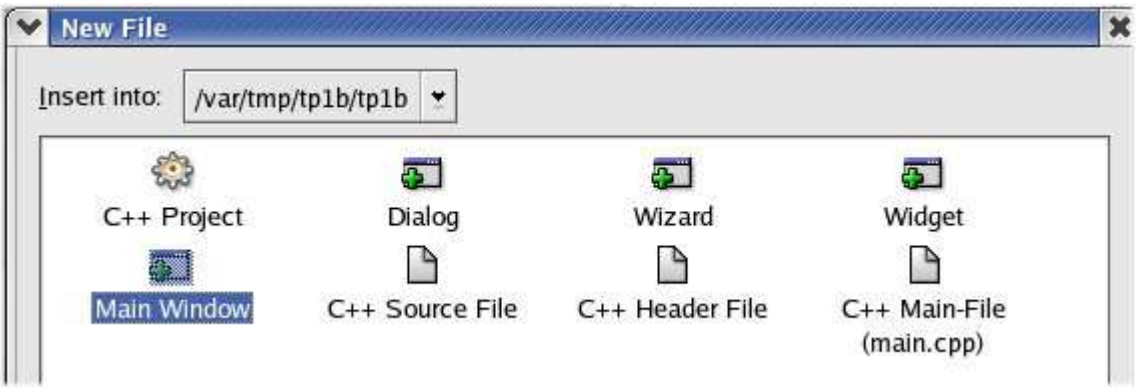
Créez un nouveau répertoire td2b et relancez qtdesigner. Choisir un projet C++ project



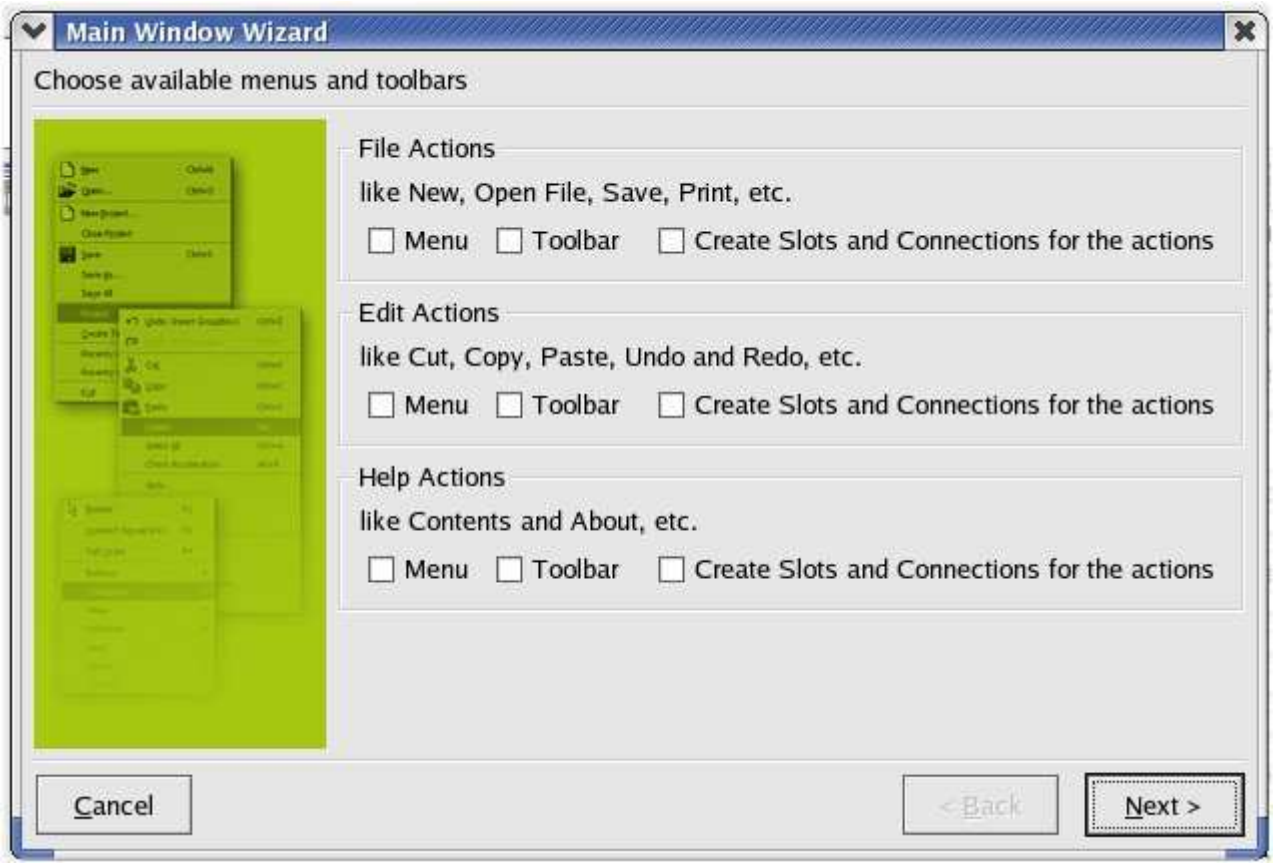
Nommez votre fichier td2b dans le répertoire td2b



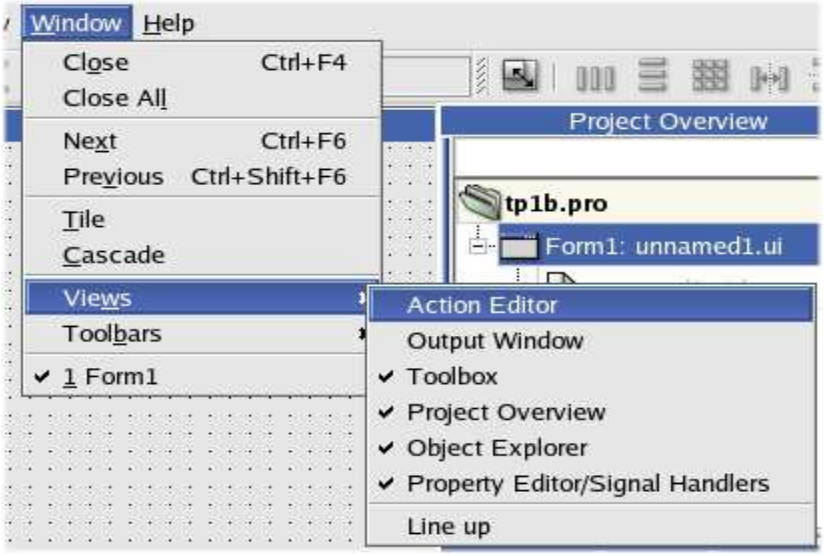
Choisir une fenêtre principale



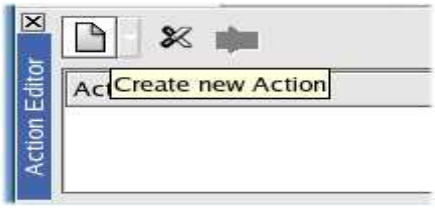
On peut utiliser des menus ou des barres d'outils préétablis



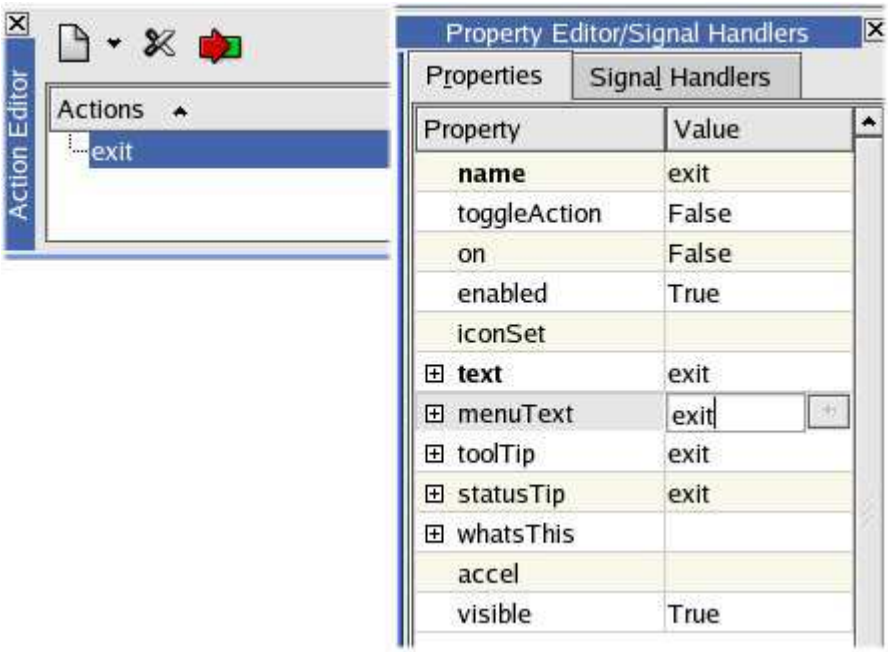
Créez une action dans l'éditeur d'action



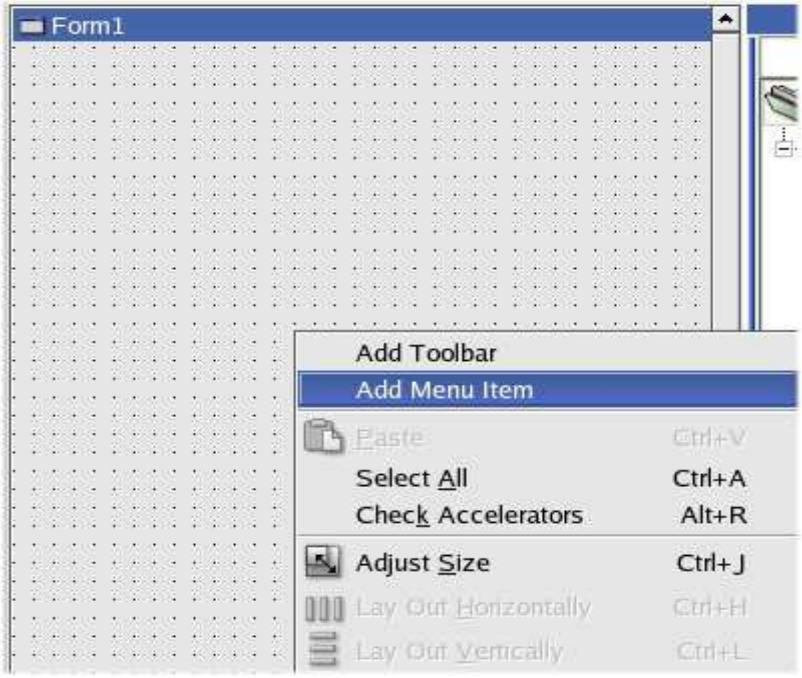
Cliquez sur create new action



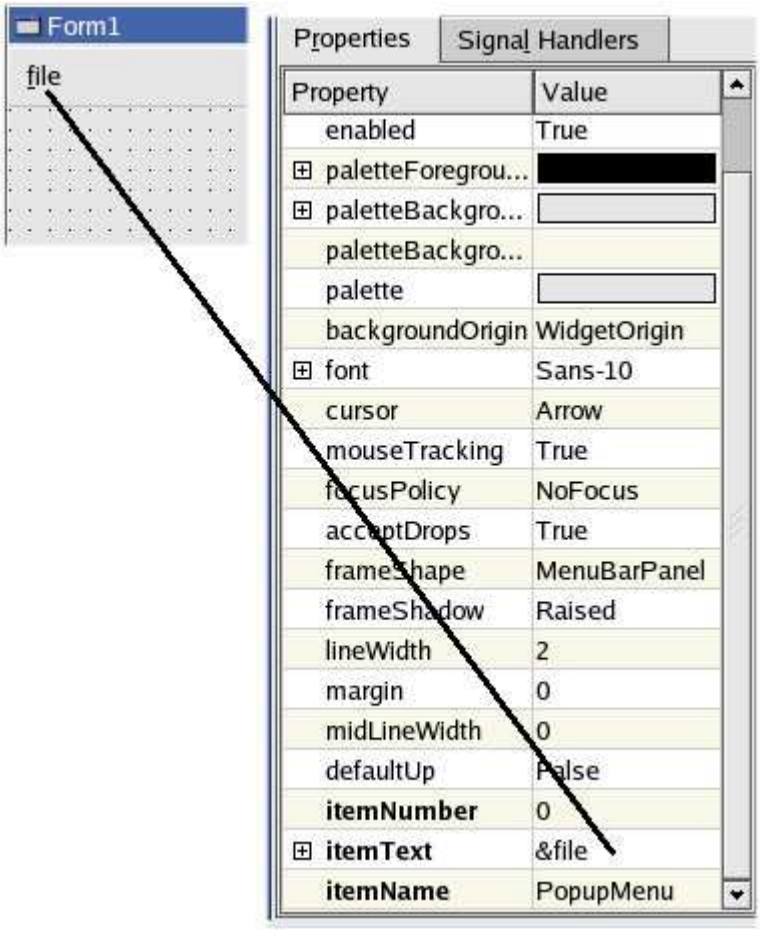
Nommez cette nouvelle action exit en double cliquant sur l'action



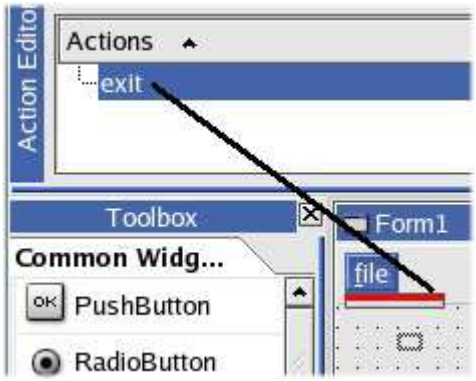
Cliquez droit sur la fenêtre Form1 et choisir add menu item



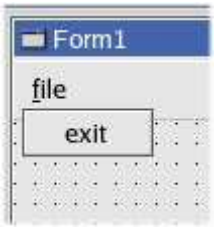
Modifiez le nom du menu en &file le & permet d'avoir un lien hypertexte



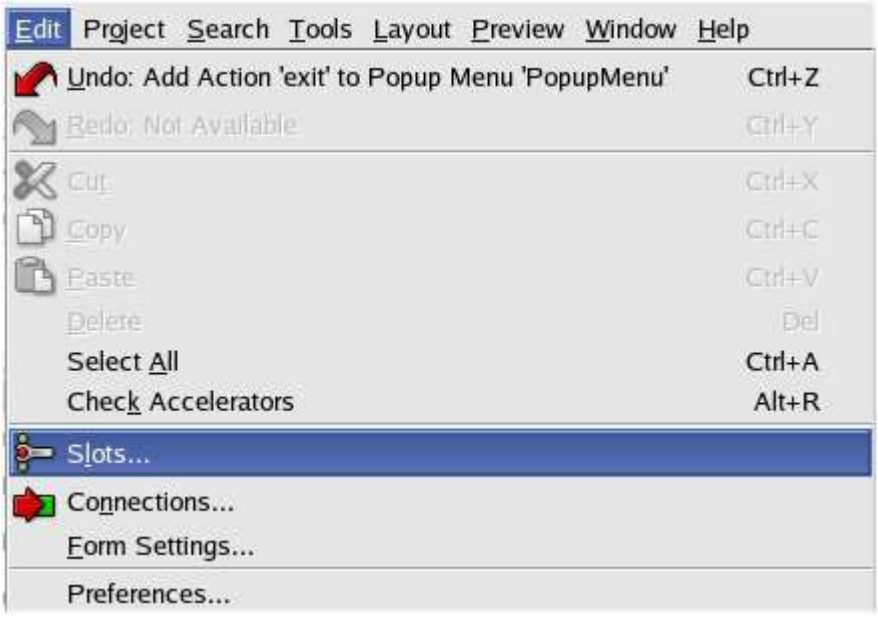
Faites glisser l'action exit sur le menu jusqu'à un trait rouge apparaisse



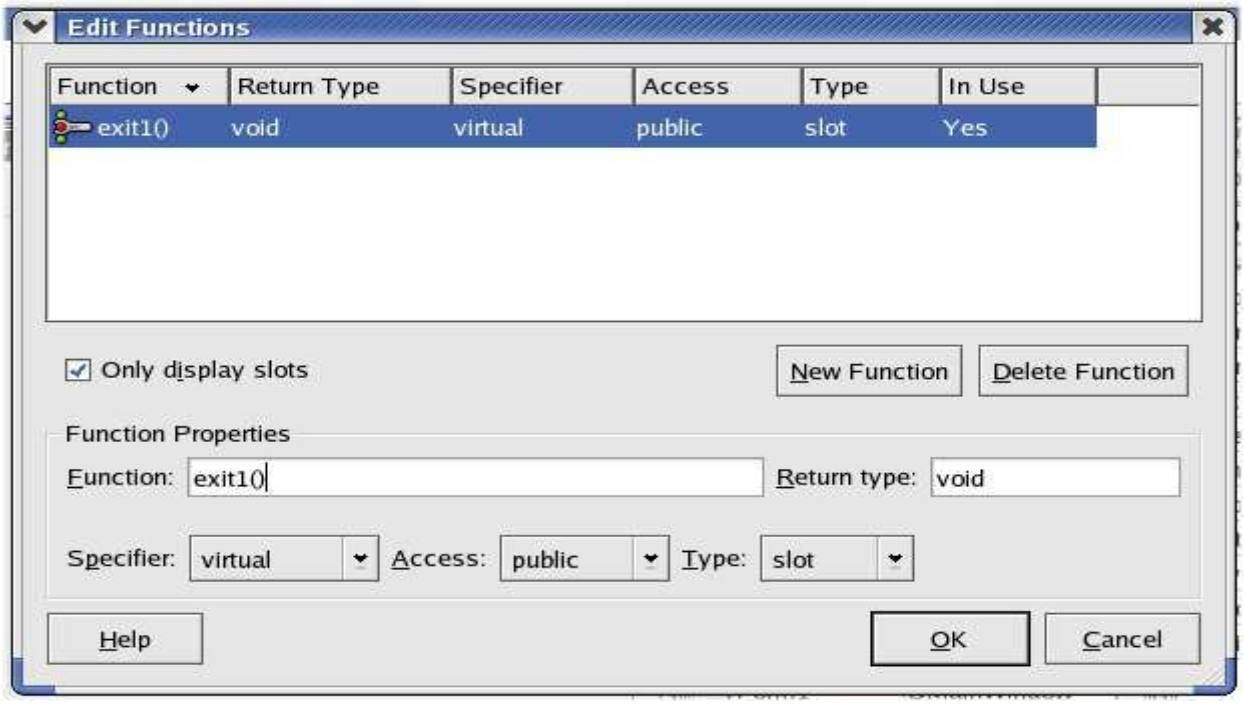
Le sous menu exit apparait



Créez un slot pour le sous menu exit



Le slot sera associé à la fonction exit1()

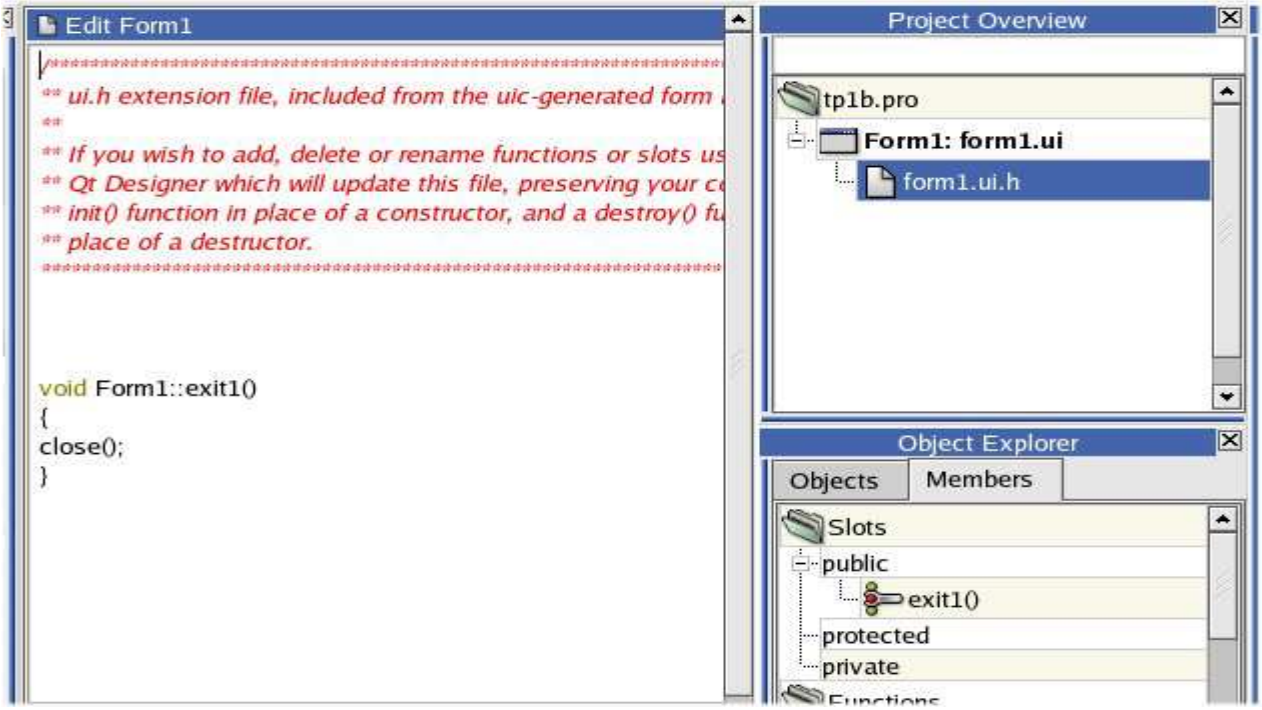


Cliquez droit sur l'action exit afin de connecter un signal au slot associé à la fonction exit1()





Cliquez sur le fichier form1.ui.h et complétez la fonction exit1()



Afin de pouvoir compiler et exécuter le programme, nous avons besoin d'un fichier main.cpp que voici:

```
#include <qapplication.h>
#include "form1.h"
int main( int argc, char **argv )
{
QApplication a( argc, argv );
Form1 *fenetre = new Form1(0);
a.setMainWidget(fenetre);
fenetre->show();
return a.exec();
}
```

✍ Réalisez sous gedit ce programme et enregistrez le sous main.cpp

Pensez à rajouter ce fichier main.cpp au projet grâce au menu "Project/Add Files…".
Lancez dans une console, ouverte dans le répertoire du projet : qmake -project && qmake && make afin de générer un fichier Makefile puis de le compiler.

✍ Testez votre programme

Quelques widgets sous QT

1) La classe QLineEdit

QLineEdit : champ de texte à une seule ligne

```
#include <qlineedit.h>
QLineEdit *zonetext=new QLineEdit("", &fenetre);
```

QLineEdit est une petite zone dans laquelle l'utilisateur peut taper un texte ou afficher du texte.



Text: (Propriété)

Pour affecter un texte à un QLineEdit, il suffit d'affecter sa propriété setText :

```
zonetext->setText("bonjour");
```

Pour lire le texte d'un QLineEdit, il suffit de lire sa propriété text:

```
QString text1 ;
text1=vitesse->text();
```

Pour transformer un QString en un entier ou en double

```
int a ;
QString chiffrestr
a= chiffrestr.toInt()
```

QString QString::number (double *n*, char *format* = 'g', int *precision* = 6) `[static]`

Retourne une chaine équivalente au double n, formaté en accord avec la précision et le format spécifié

Format	Meaning
e	format as [-]9.9e[+|-]999
E	format as [-]9.9E[+|-]999
f	format as [-]9.9
g	use e or f format, whichever is the most concise
G	use E or f format, whichever is the most concise

Pour transformer une chaine de caractère chiffre en un entier ou en double

```
QString QString::number ( long n, int base = 10 )
long a = 63;
QString s = QString::number(a, 16);           // s == "3f"
QString t = QString::number(a, 16).toUpper();     // t == "3F"
```

double QLocale::toDouble (const QString & *s*, bool * *ok* = 0) const

Retourne un paramètre double à partir de la chaine *s*, ou 0.0 si la conversion a échouée.

```
bool ok;
 double d;
d = c.toDouble( "1234.56", &ok );  // ok == true, d == 1234.56
 d = c.toDouble( "1,234.56", &ok ); // ok == true, d == 1234.56
 d = c.toDouble( "1234,56", &ok );  // ok == false
```

2) *Le widget* QLCDNumber

Le widget QLCDNumber affiche un nombre avec des digits comme un afficheur LCD.

```
#include <qlcdnumber.h>
```



Héritage: QFrame.

Membres privés

digitCount : int
intValue : int
mode : Mode
numDigits : int (deprecated)
segmentStyle : SegmentStyle
smallDecimalPoint : bool
value : double

Méthodes public

QLCDNumber (QWidget * *parent* = 0)
QLCDNumber (uint *numDigits*, QWidget * *parent* = 0)
~QLCDNumber ()
bool **checkOverflow** (double *num*) const
bool **checkOverflow** (int *num*) const
int **digitCount** () const
int **intValue** () const
Mode **mode** () const
SegmentStyle **segmentStyle** () const
void **setDigitCount** (int *numDigits*)
void **setMode** (Mode)
void **setSegmentStyle** (SegmentStyle)
bool **smallDecimalPoint** () const
double **value** () const

Signals

void **overflow** ()

Slots publiques

void **display** (const QString & *s*)
void **display** (double *num*)
void **display** (int *num*)
void **setBinMode** ()
void **setDecMode** ()
void **setHexMode** ()
void **setOctMode** ()
void **setSmallDecimalPoint** (bool)

3) *La classe* QPushButton

Le widget QPushButton fournit un bouton de commande




```
#include <qpushbutton.h>
```

Héritage: QAbstractButton.

Public Functions

	QPushButton (QWidget * *parent* = 0)
	QPushButton (const QString & *text*, QWidget * *parent* = 0)
	QPushButton (const QIcon & *icon*, const QString & *text*, QWidget * *parent* = 0)
	~QPushButton ()
bool	**autoDefault** () const
bool	**isDefault** () const
bool	**isFlat** () const
QMenu *	**menu** () const
void	**setAutoDefault** (bool)
void	**setDefault** (bool)
void	**setFlat** (bool)
void	**setMenu** (QMenu * *menu*)

Fonctions protégées réimplantées

virtual bool **event** (QEvent * *e*)
virtual void **focusInEvent** (QFocusEvent * *e*)
virtual void **focusOutEvent** (QFocusEvent * *e*)
virtual bool **hitButton** (const QPoint & *pos*) const
virtual void **keyPressEvent** (QKeyEvent * *e*)
virtual void **paintEvent** (QPaintEvent *)

TDN°3 Dialogue avec le simulateur

1) But

Mettre en œuvre un échange d'information entre un PC et le simulateur via un réseau ETHERNET

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

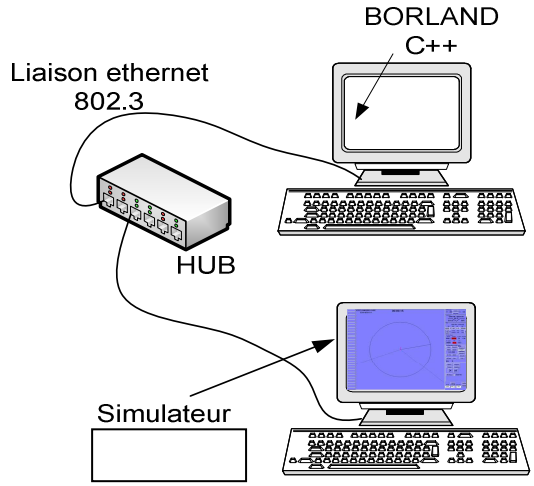
-D'exploiter la documentation sur les composants VCL afin de développer un applicatif permettant l'échange d'information entre un serveur (simulateur) et un client (PC)

-D'analyser grâce à un analyseur de trame les échanges entre le client et le serveur

3) Prérequis

Cours sur les réseaux à technologie IP et le langage C++.

4) Schéma de câblage



5) Présentation

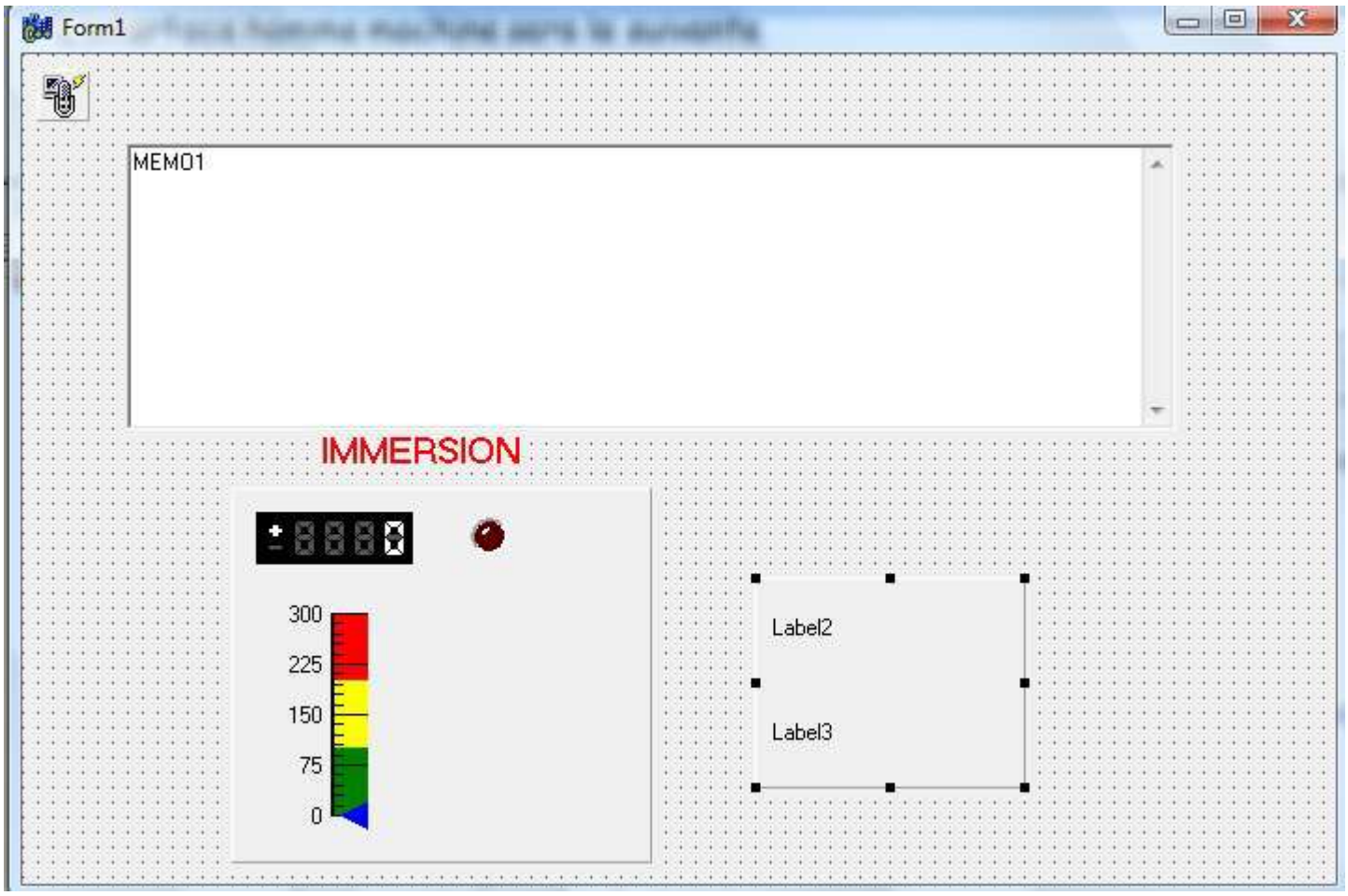
Dans Ces multiples capacités de communication, le simulateur dispose entre autre de 2 types de serveur :

- Un serveur lui permettant d'envoyer périodiquement les informations sur la navigation. Ces informations sont envoyées sous le protocole UDP en diffusion générale pour le sous réseau. Le numéro de port destination est le.
- Un deuxième serveur lui permettant de recueillir certaines informations issues du réseau navigation

6) Exploitation des données envoyées par le calculateur

On souhaite récupérer le datagramme envoyé par le simulateur en mode UDP et exploiter l'information immersion.

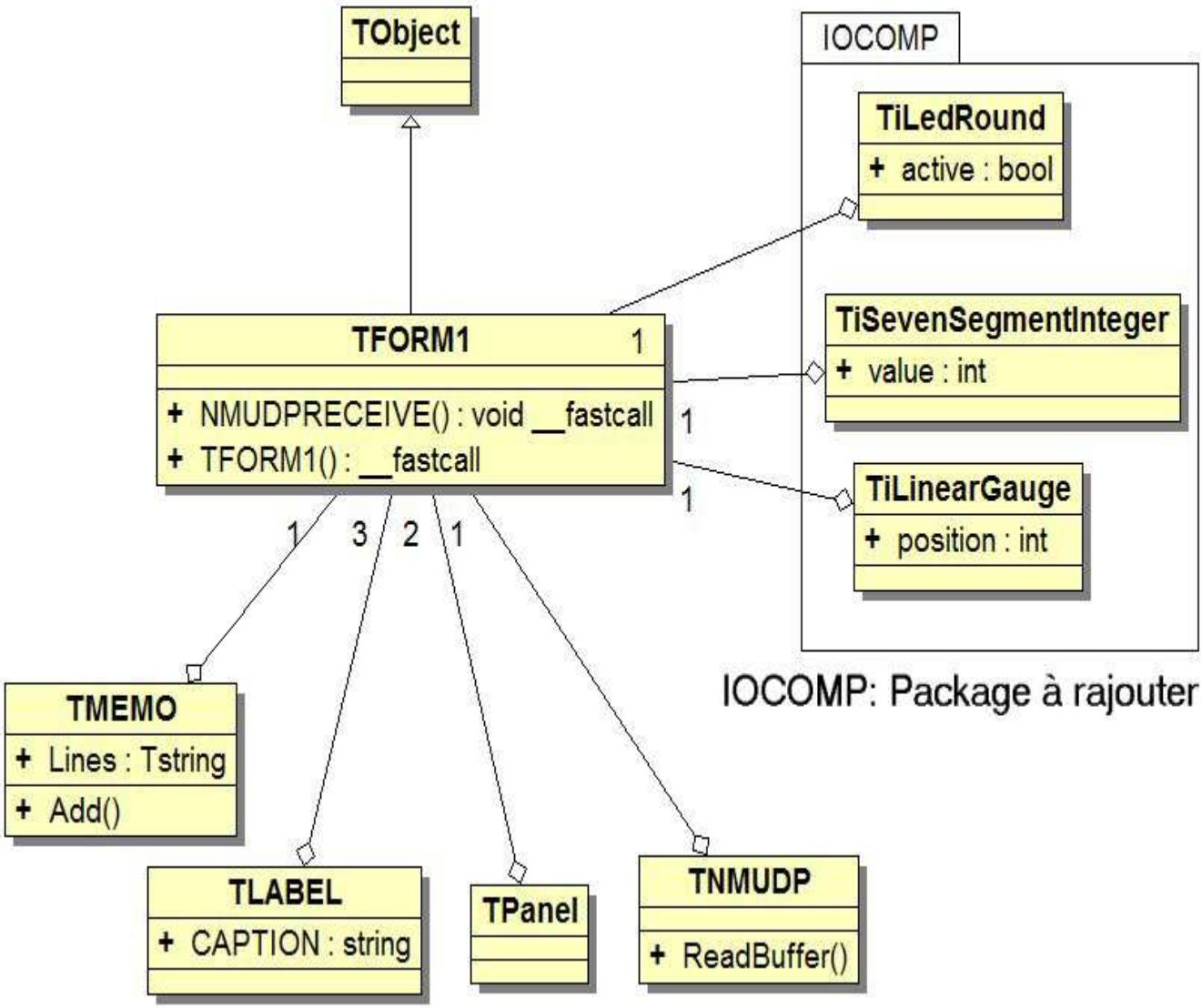
L'interface homme machine sera la suivante :



2 composants graphiques nous permettent de visualiser l'information immersion, un afficheur 7 segments et une jauge linéaire. La LED sera allumée dans le cas ou l'immersion est supérieure à 300m.

Tous les composants nécessaires sont décrits dans le dossier Page 79 et 88.

6.1) Diagramme de classes



Pour récupérer les datagrammes UDP nous utiliserons la classe TNMUDP. On lancera la méthode NMUDPRECEIVE lorsqu' apparaitra l'événement OnDataReceive (voir l'inspecteur d'objet de l'objet NMUDP1). On récupérera les données du datagramme UDP grâce à la méthode ReadBuffer (voir dossier annexe). Les octets reçus sous forme de caractère ASCII seront rangés dans un tableau nommé Buff et la valeur de l'immersion se trouve :

- Pour les centaines dans Buff[NumberBytes-15] ou NumberBytes est le nombre d'octets reçus
- Pour les dizaines dans Buff[NumberBytes-14]
- Pour les unités dans Buff[NumberBytes-13]

6.2) Construction de l'applicatif

6.2.1) Analysez grâce à un analyseur de trame (wireshark) les échanges entre le serveur et le client. Identifiez la trame de données et donnez la valeur du numéro de port de destination.

6.2.2) Réalisez l'interface graphique, en plaçant les différents objets sur la feuille form1.

Les onglets pour choisir les composants sont les suivants :

- Iocomp Std
- Fastnet
- Standard

Modifiez ci-nécessaire dans l'inspecteur d'objets, les propriétés ou les événements de chacun des objets.

6.2.3) On donne le code à compléter pour le constructeur de l'objet form1 (Document réponse N°1), complétez celui-ci afin de visualiser l'adresse IP de l'ordinateur ainsi que le nom Netbios (nom de l'ordinateur).

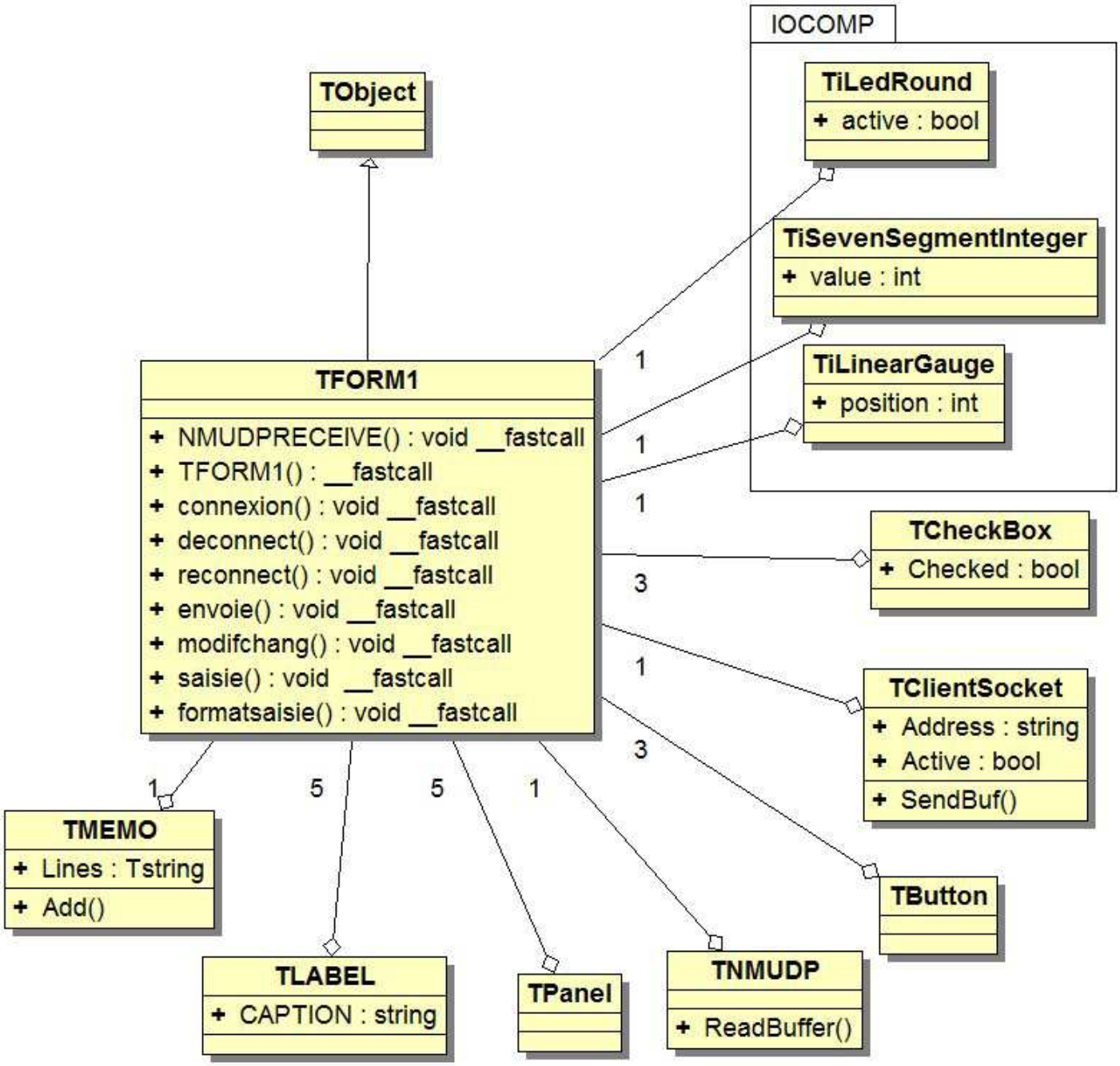
6.2.4) Donnez le code de la méthode NMUDPRECEIVE afin de réaliser les opérations suivantes :

- Récupérez dans un tableau Buff les données du datagramme IP.

-Affichez grâce à l'objet MEMO1, chaque datagramme sur une ligne sous le format indiqué page suivante

- En isolant l'information immersion dans le tableau Buff, calculez celle-ci sous la forme d'un entier result.

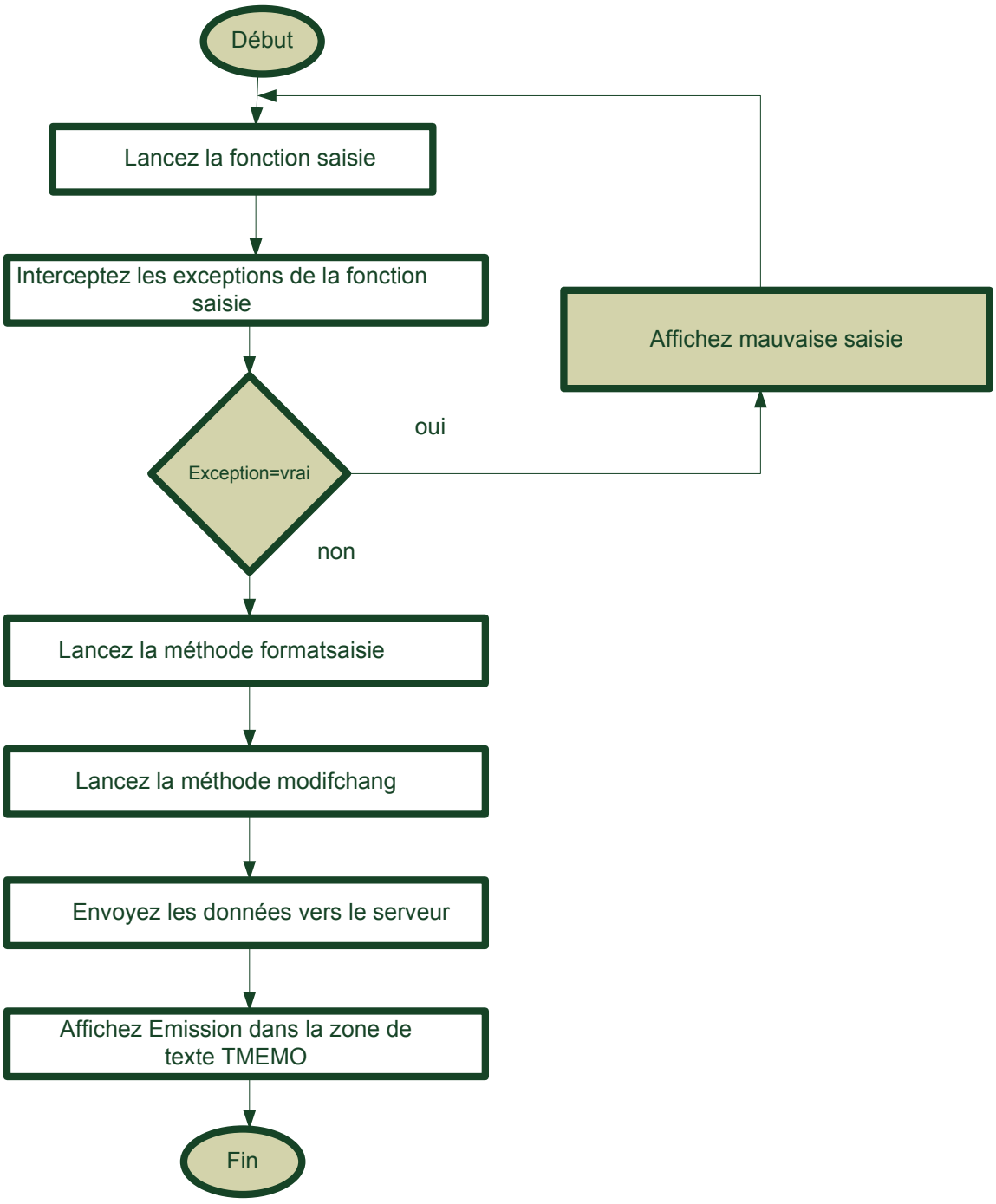
7.1) Diagramme de classes

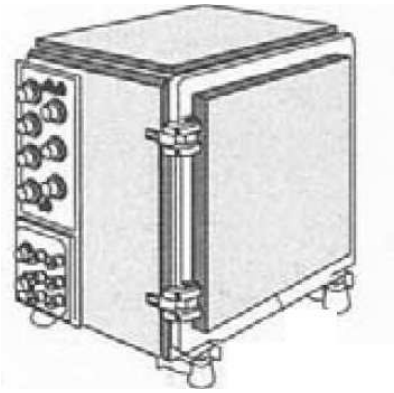


Document réponse N°1

```
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
WORD wVersionRequested;
WSADATA wsaData;
char name[255];
AnsiString ip;
PHOSTENT hostinfo;
nav1= new nav();
P1= new nav2 ;
wVersionRequested = MAKEWORD( 2, 0 );
if ( WSAStartup( wVersionRequested, &wsaData ) == 0 )
{
if( gethostname ( name, sizeof(name)) == 0)
{
_____; //affichage nom de l'ordinateur (Tlabel)
if((hostinfo = gethostbyname(name)) != NULL)
{
ip = inet_ntoa (*(in_addr *)*hostinfo->h_addr_list);
_____; //affichage adresse IP(Tlabel)
}
}
WSACleanup( );
}
}
```

Algorigramme de la méthode envoie





TD N°4 : Borland C++ Builder

(Développement D'un simulateur pour)

TDN°4 Développement d'un simulateur

1) But

Mettre en œuvre un échange d'information entre un simulateur et le simulateur via un réseau ETHERNET

2) Objectifs : L'étudiant devra être capable :

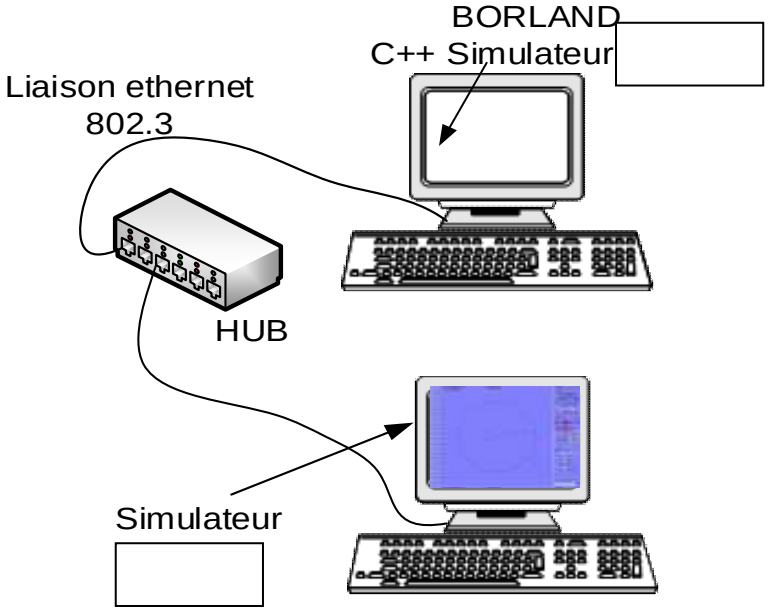
-D'exploiter la documentation sur les composants VCL afin de développer un applicatif permettant l'échange d'information entre _____et le simulateur

-D'analyser grâce à un analyseur de trame les échanges entre un client et un serveur

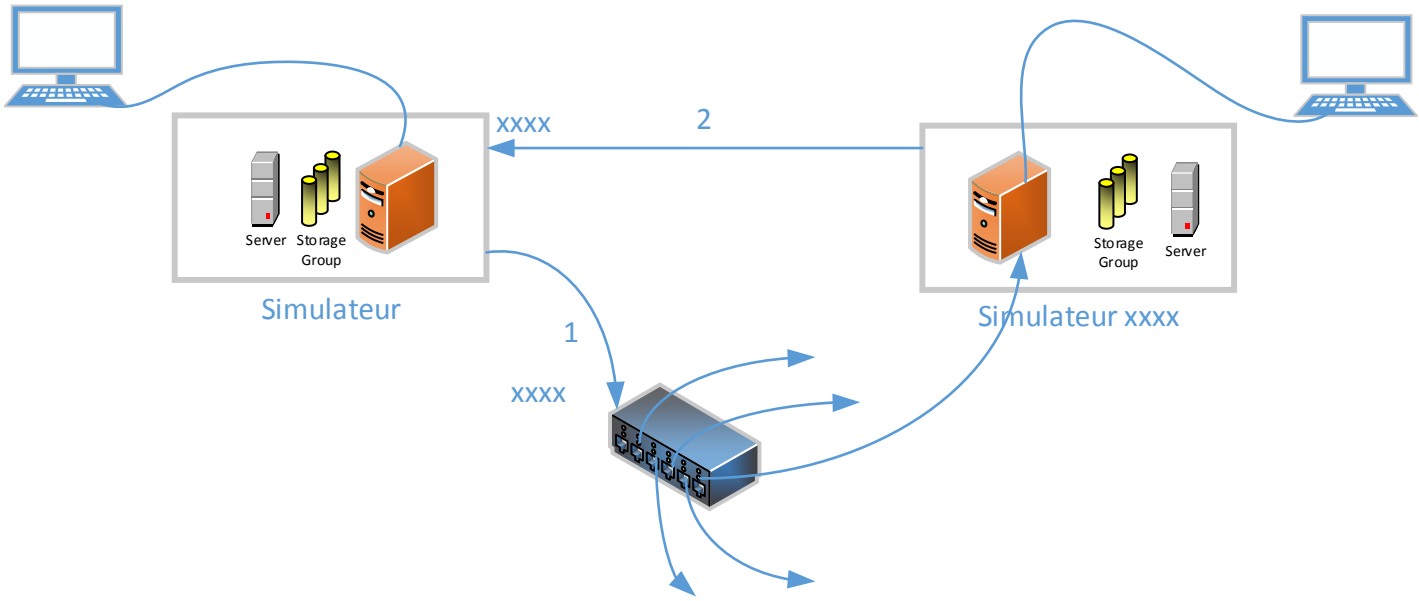
3) Prérequis

Cours sur les réseaux à technologie IP et le langage C++.

4) Schéma de câblage



4) Echange de trames entre le simulateur et le simulateur



Message sortant du simulateur

Le simulateur envoie périodiquement les informations par le simulateur. Ces trames sont notées 1 sur le schéma et sont constituées :

@ IP	Mode	Port destination	Informations
192.9.200.255	UDP	xxxx	mod/

D'autres trames plus complètes sont envoyées à la demande du simulateur. Elle comporte les principales caractéristiques du simulateur.

@ IP	Mode	Port destination	Informations
192.9.200.255	UDP	xxxx	Charg charg/

Message sortant du simulateur

Ces messages sont notés 2 sur le schéma

Le premier message est une demande de recharge des _____ de la base de données du _____

@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxxx	Recharg

Le deuxième message est

@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxxx	

Le troisième message est

@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxx	

Le quatrième message est

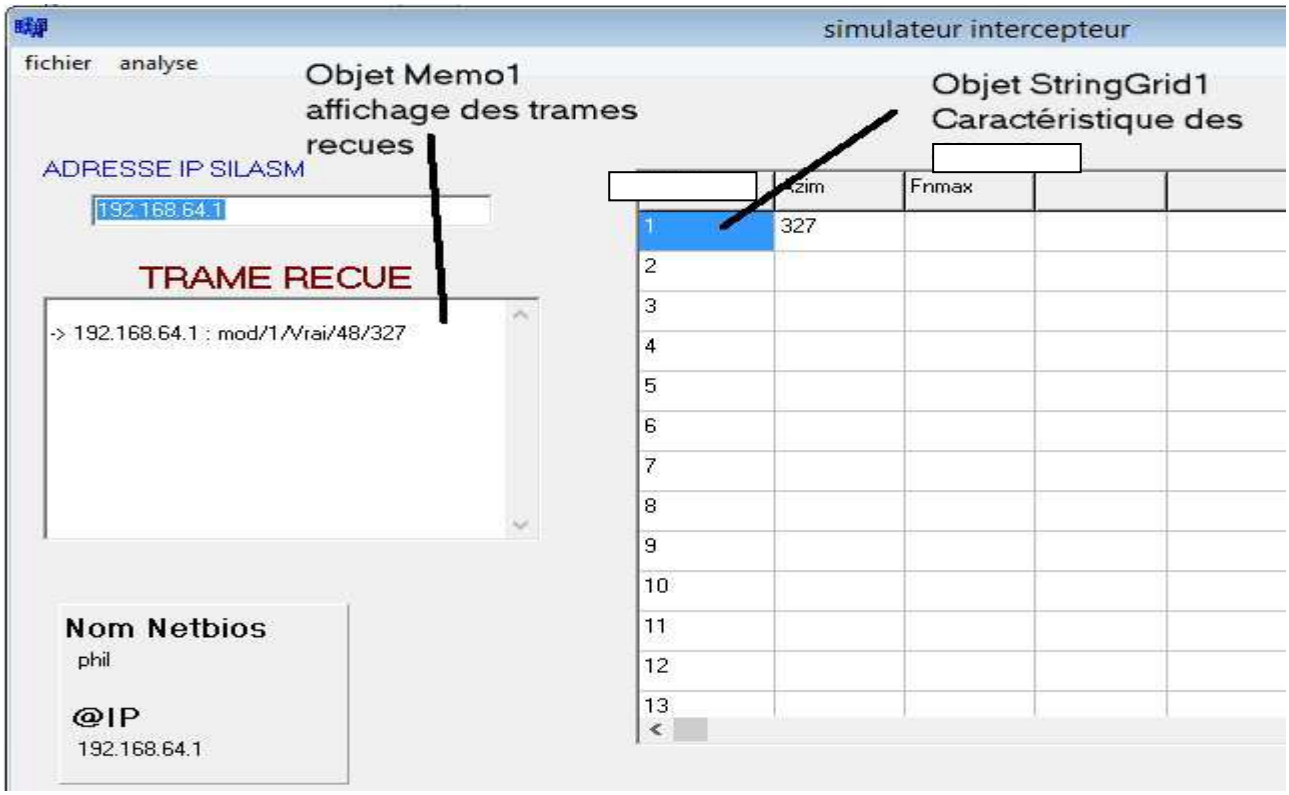
@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxx	

Le cinquième message est

@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxx	

5) Exploitation des données envoyées par le simulateur

L'interface homme machine que l'on souhaite réaliser dans un premier temps est la suivante :



L'objet Memo1 permettra l'affichage des trames reçues. Ces trames auront comme port destination le port xxxx. L'objet StrinGrid1 caractérisera certains paramètres des . Etant donné le nombre important de paramètres qui caractérisent , seul l'azimuth et le paramètre Fnmax sera prise en compte respectivement dans la colonne N°2 et N°3.

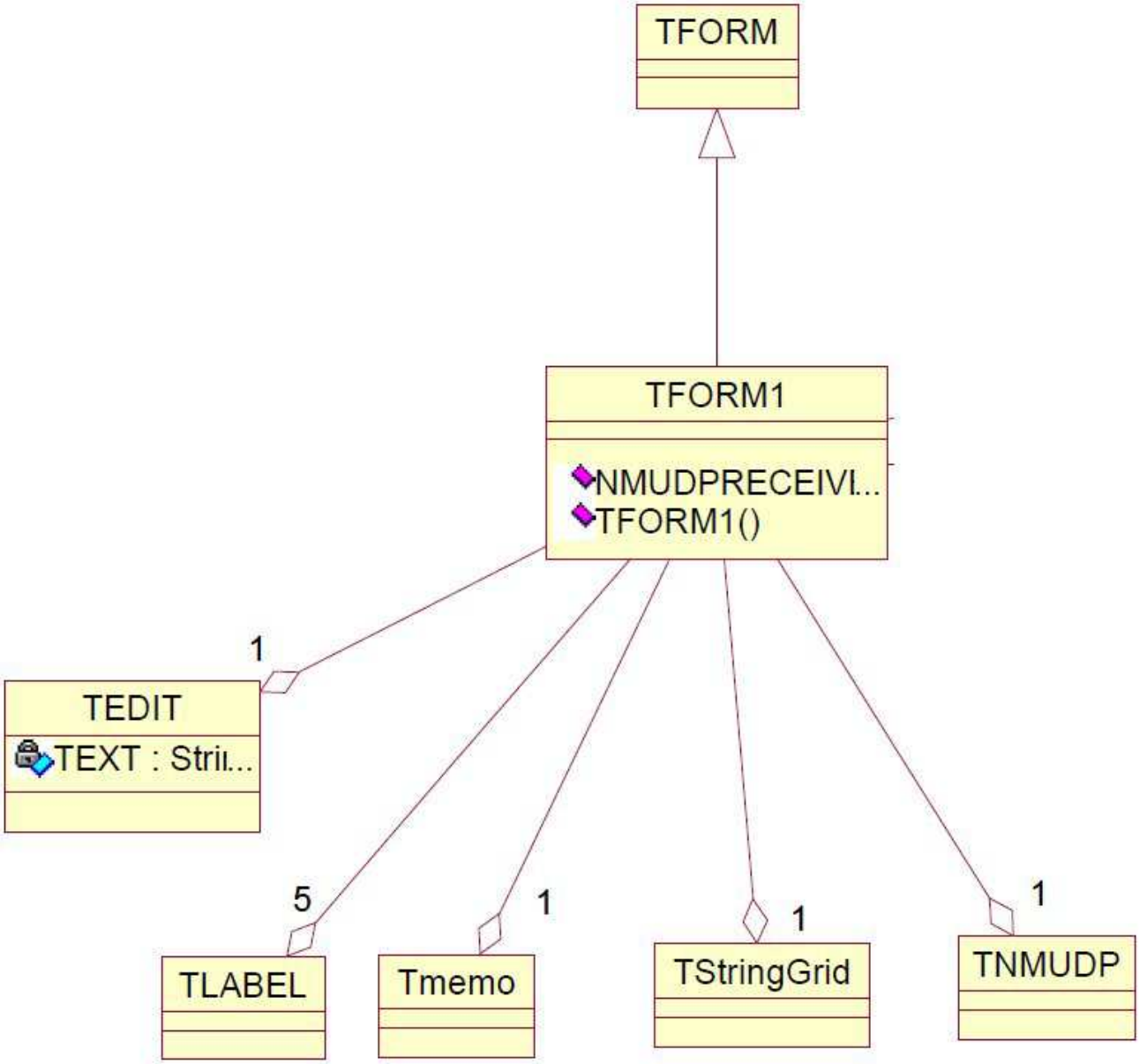
Comme pour le TDN°3 on utilisera la classe TNMUDP. On lancera la méthode NMUDPRECEIVE lorsqu' apparaitra l'événement OnDataReceive (voir l'inspecteur d'objet de l'objet NMUDP1). On récupérera les données du datagramme UDP sur le port destination xxxx grâce à la méthode ReadBuffer (voir dossier annexe). Les octets reçus sous forme de caractère ASCII seront rangés dans un tableau nommé Buff

5.1) En vous aidant du diagramme de classe page suivante, Réalisez l'interface homme machine

5.2) En vous inspirant du TDN°3 affichez sur l'objet TMEMO les informations qui circulent en mode UDP (port destination xxxx).

5.3) Dans la méthode constructeur de la classe TFORM, créez l'objet StringGrid1 et initialisez ce tableau avec les paramètres suivants :

- Nombre de ligne 49
- Nombre de colonne 7
- Initialisation des champs colonne 1 avec xxxxxx, colonne 2 Azim, Colonne 3 Fnmax et le numéro de 1 à 48



5.4) On donne ci-dessous le code de la méthode NMUDPRECEIVE qui permet d'exploiter la trame du type mod voir page N°56 et de mettre à jour le tableau avec le numéro et son paramètre azimuth

```
void __fastcall TForm1::NMUDPRECEIVE(TComponent *Sender, int NumberBytes,
    AnsiString FromIP, int Port)
{ int result, nbrslash=0, numero=0, indicetab;
char adress[3];
char  azimuth[4];
char Buff[1000];
char freq[5];
```

```
// on reçois des données donc on les lis
NMUDP1->ReadBuffer(Buff, NumberBytes,NumberBytes);
indicetab=0;

/* extraction de l'information numéro de radar et azimuth de la trame du type mod */

if ((Buff[0]=='m')&&(Buff[1]=='o')&&(Buff[2]=='d')){
for (int i=0; i< NumberBytes ; i++)
{
  if   (Buff[i]=='/')  nbrslash++;
  switch (nbrslash){
  case  1: adress[indicetab++]=Buff[i+1]; break; // récupération du numéro du radar
  case  4:  azimuth[indicetab++]=Buff[i+1]; break; // récupération de la valeur de l' azimuth
  default: indicetab=0;
 }

}
// et on les affiches
adress[2]=0x00;
azimuth[3]=0x00;
if (adress[1]=='/') adress[1]=0x00; // Numéro du radar codé sur un seul chiffre
if (azimuth[2]=='/') azimuth[2]=0x00;//  Azimuth codé sur 2 chiffres
if (azimuth[1]=='/') azimuth[1]=0x00; ;//  Azimuth codé sur 1 chiffres
MEMO1->Lines->Add("-> "+FromIP+" : "+Buff);
AnsiString Chaine1 ;;
AnsiString Chaine2 ;
AnsiString nradar ;
Chaine1=AnsiString(adress);
Chaine2=AnsiString(azimuth);
 indicetab=StrToInt(Chaine1);
StringGrid1->Cells[1][indicetab] = Chaine2;// Mise à jour dans le tableau
}
```

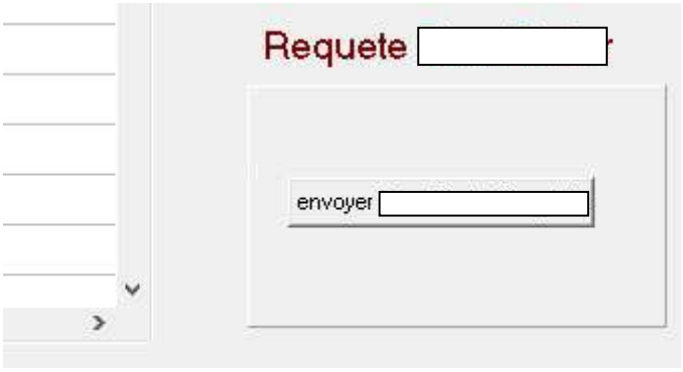
Donnez les lignes de code qui permettent d'exploiter la trame du type et de mettre à jour le Numéro avec les paramètres azimuth et Fnmax. Testez votre programme en utilisant le programme

5.5) Demande de rechargement de la base de donnée

Le simulateur peut demander au simulateur un rechargement de la base de données. La requête envoyée par le simulateur est la suivante :

@ IP	Mode	Port destination	Informations
Serveur	UDP	xxxx	Recharg

On rajoutera dans l'IHM le bouton suivant :

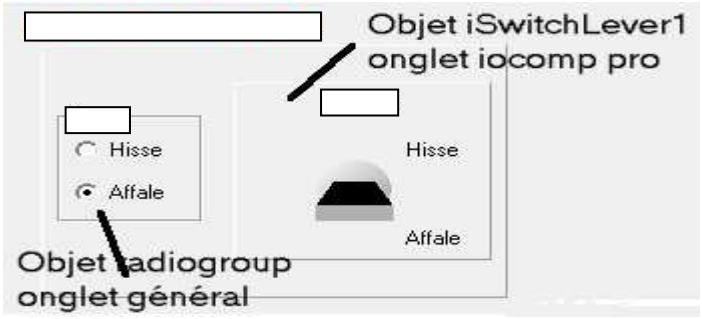


On gérera l'événement on click du bouton afin d'envoyer la chaine de caractère Recharg. On créera un deuxième objet NMUDP2 de la classe TNMUDP afin d'envoyer grâce à la méthode SendBuffer la chaine de caractère précédente.

A partir des Recommandations précédentes mettez en place cette demande de rechargement de la base de données

5.6) Gestion de

Le simulateur peut communiquer au simulateur via simu l'état de
En vous aidant des trames générées par le simulateur page N°57, proposez les lignes de code qui nous permettent de communiquer. On se servira des objets suivants :

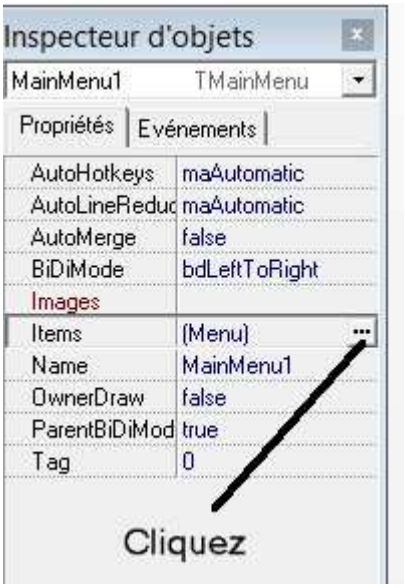


5.7) affichage sur un graphe azimuth fréquence

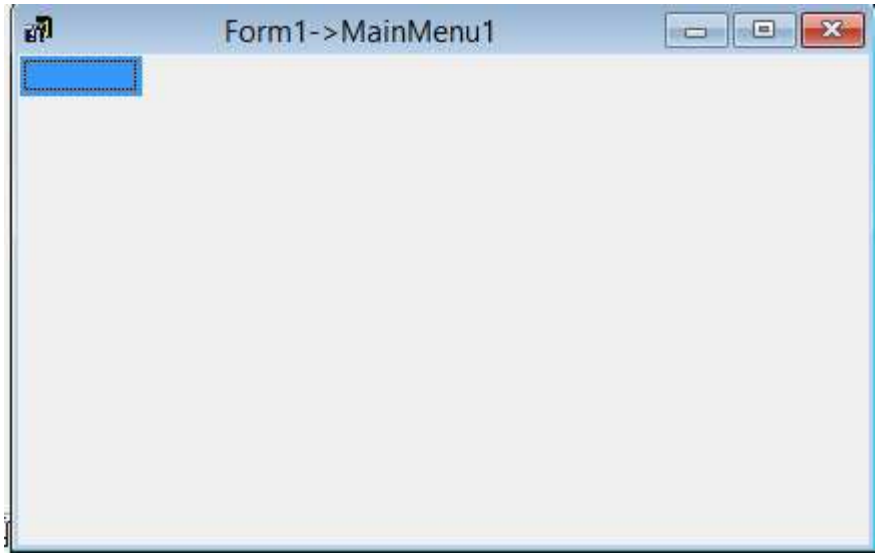
En premier lieu, on va créer un menu. Sous borland C++ l'objet de la classe TMainMenu nous permet d'automatiser la création de ce menu.
Insérez cet objet sous notre feuille



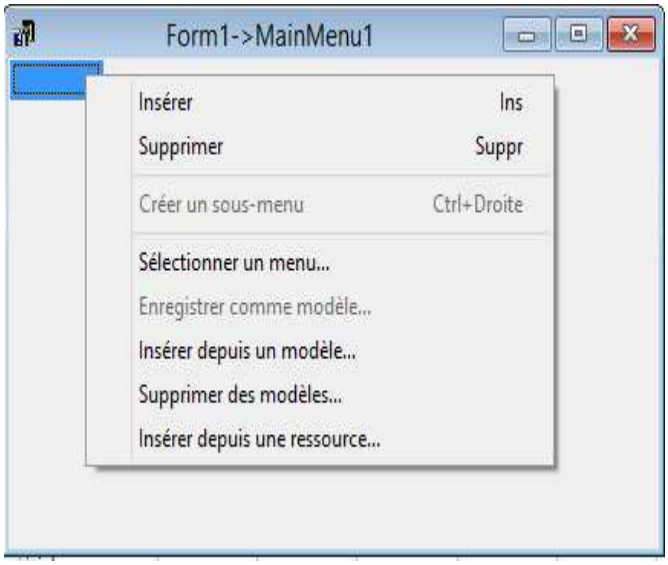
Onglet général



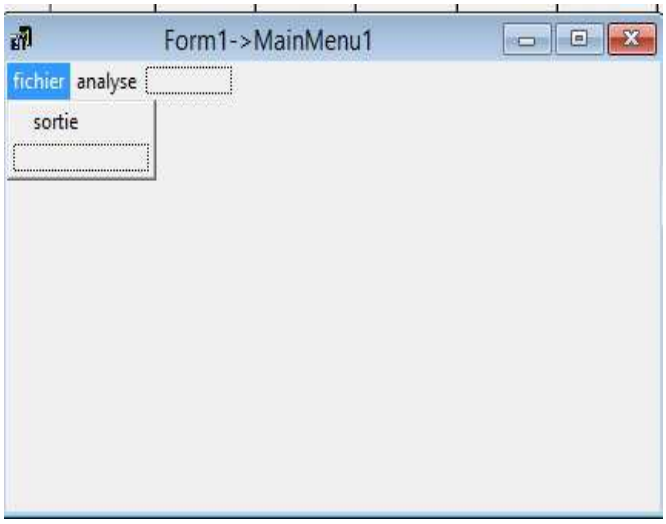
Le menu apparait



Cliquez droit pour créer des menus et des sous menus



On obtient le menu suivant :

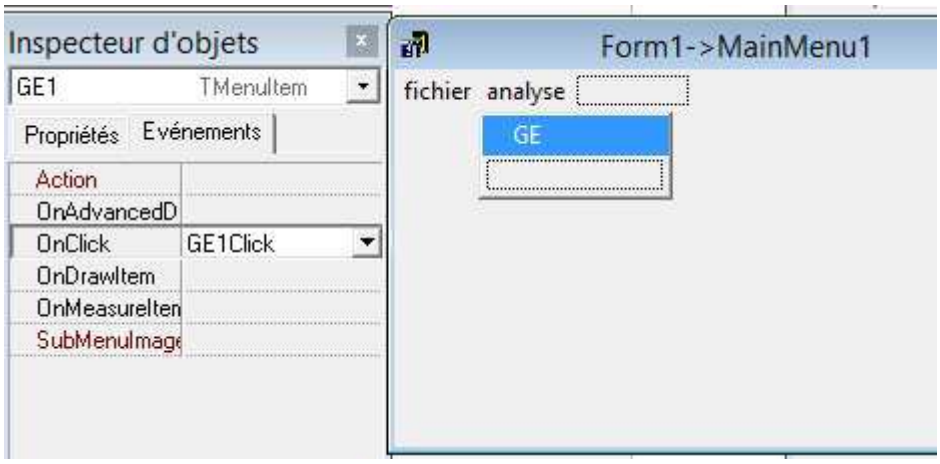




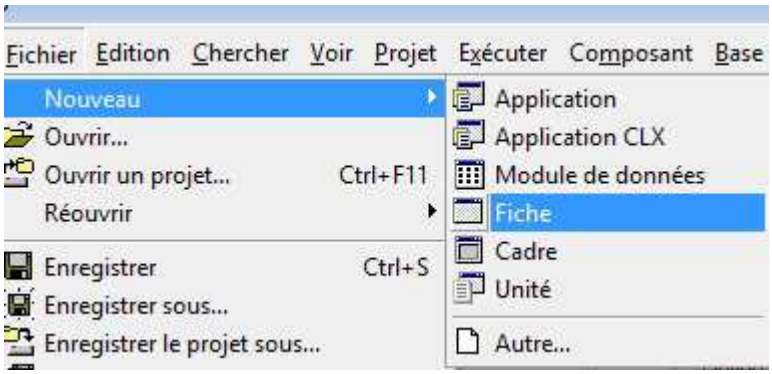
Dans l'inspecteur d'objets, créez la méthode fermer associée à l'événement OnClick.
Complétez le code suivant pour la méthode fermer :

```
void __fastcall TForm1::fermer(TObject *Sender)
{
Close();  // méthode permettant de fermer le programme
}
```

Dans le sous menu GE du menu analyse, créez la méthode GE1click associé à l'événement onclick



On va créer une nouvelle fiche afin d'afficher les résultats de l'analyse GE et d'afficher sur un graphe azimut fréquence les différents éléments:
Allez dans :

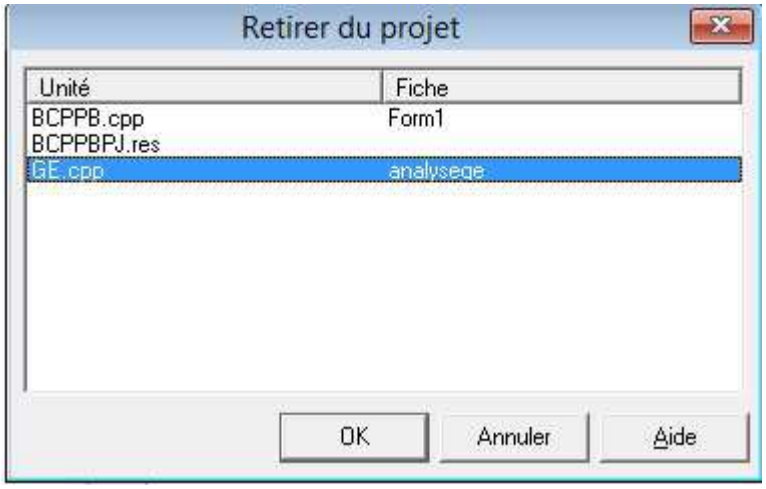


Une nouvelle fiche apparait. Changez le nom de la fiche en analysege.

On voit apparaitre le code source associé à ce nouvel objet de la classe TFORM
Unit2.cpp et unit2.h dans le gestionnaire de projet

Renommez le nom des fichiers sources dans
Fichier → enregistrez sous → GE.cpp le fichier d'en tête est aussi modifié en GE.h

Dans retirer du projet on voit apparaitre le code source GE.cpp et sa fiche associée :

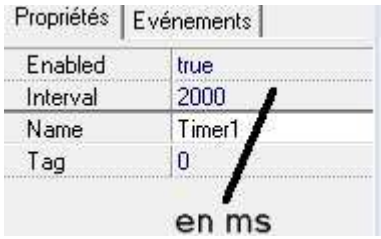


Pour voir la fiche allez dans Voir →fiche et sélectionnez analysege



Insérez l'objet iXYplot dans le fiche analyse et enregistrez votre projet
Dans la méthode GE1click de la classe TFORM1 rajoutez le code suivant :
analysege->Show(); afin de faire apparaitre cette fiche suite à l'événement on click du sous menu Ge .

Modifiez cet objet afin de changer l'axe des abscisses des ordonnées et le titre

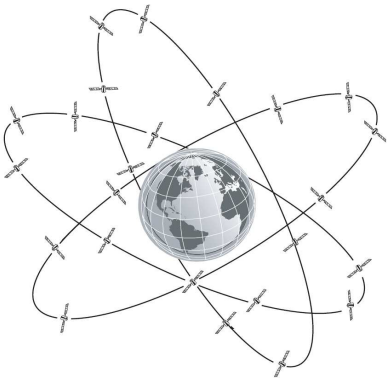


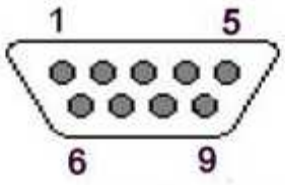
Le membre interval règle la périodicité

On donne ci-dessous le code à compléter de la méthode Timer1Timer :

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
int tabge[48][2],k;
for (int j=0;j<48;j++){
 tabge[j][0]=0;
 tabge[j][1]=0;
}
for (k=0;k<49;k++){
if(((StringGrid1->Cells[1][k+1]).Length()!=0)&&((_____________________________________)) {
tabge[k][0]=StrToInt(_____________________________________ ; // azim
tabge[k][1]=StrToInt(_____________________________________) ;  // fréquence
}
}
analysege->envoiege(_____________________________ );
}
```

Testez votre programme avec simu





TD N°5 : Borland C++ Builder

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

TD N°5 :Développement d'une IHM pour la gestion du port série

1. But

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

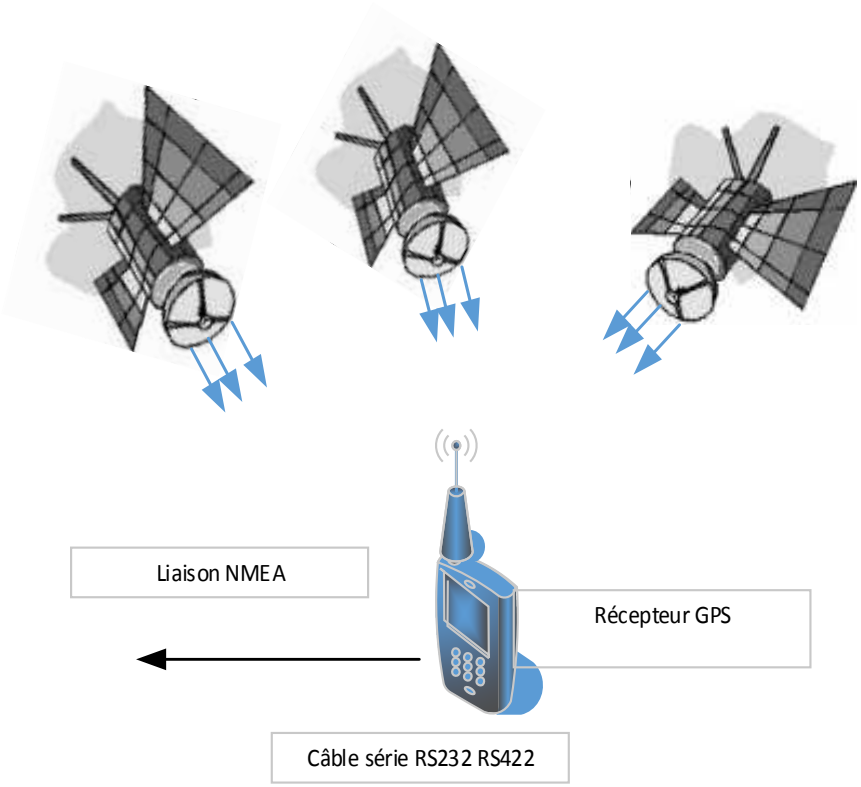
2. Objectifs : L'étudiant devra être capable de :

-D'exploiter la documentation sur les composants VCL afin de développer un applicatif permettant la gestion du port série

3. Prérequis

- Cours sur les transmissions séries
- Cours sur le langage C++.

4. Schéma de câblage



5. Présentation

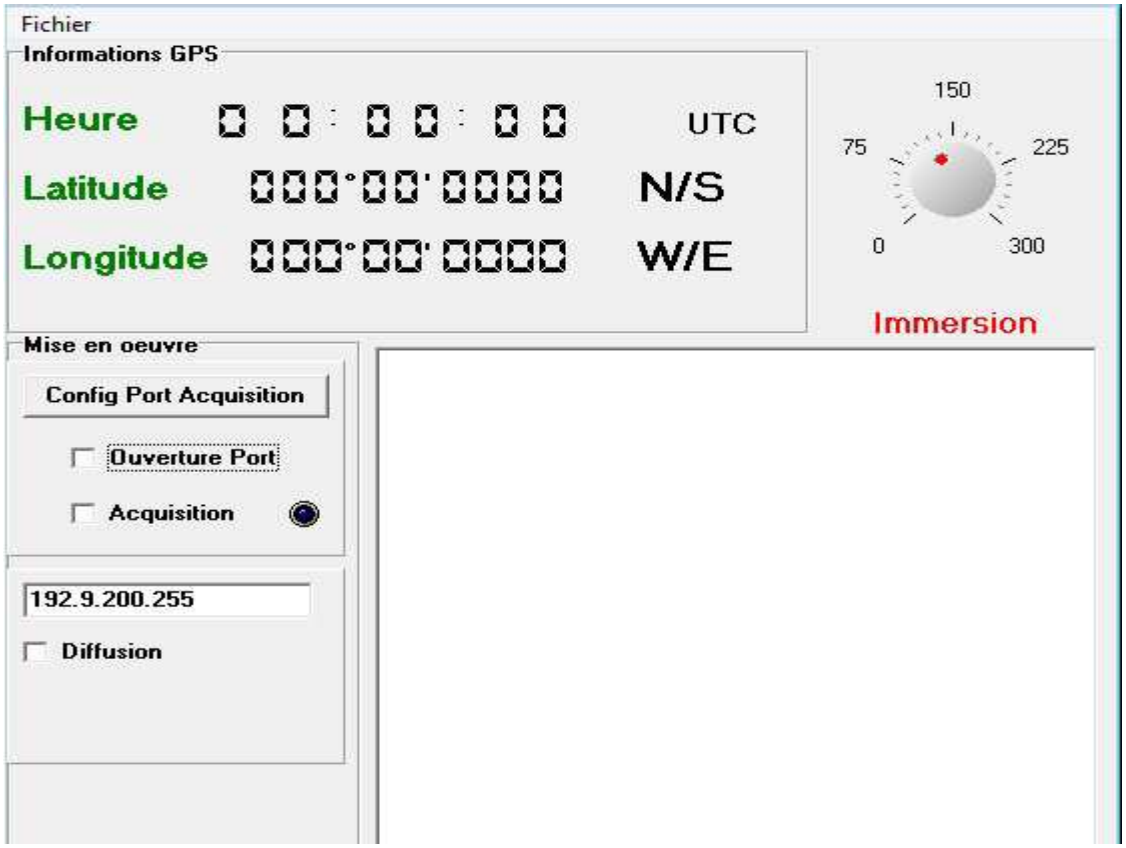
La navigation par GPS est de plus en plus importante dans le domaine maritime. On se propose lors de cette manipulation de récupérer les informations issues d'un récepteur GPS. On donne ci-dessous le schéma de principe de cette liaison. Ce TP doit nous permettre de dialoguer avec un récepteur GPS en utilisant le port série. Le composant et son paquetage TCOMPORT doit être au préalable installé dans Borland C++ Builder afin de pouvoir bénéficier des différentes méthodes nous permettant de recueillir la trame GPS.

On donne dans le dossier dénommé les composants de la bibliothèque VCL les différentes méthodes de la classe TCOMPORT

6. Gestion du port série

La mise en place du port série sera grandement facilitée grâce à la classe TCOMPORT. Cette classe décrite dans le fascicule composants de la VCL nous fournit les objets essentiels pour la gestion du port série.

L'interface homme machine sera la suivante :



La LED de la classe TCOMLED, nous indiquera si le port est ouvert :

Aucune ligne de code supplémentaire est nécessaire pour les LEDS, il faudra simplement bien paramétrée l'inspecteur d'objet pour celles-ci

Le port série dans l'inspecteur d'objet sera paramétré de la façon suivante :

- Aucun contrôle de flux
- Bit de parité aucun
- 8bits de données, 9600bauds
- 1 bit de stop

La checkbox acquisition permettra de capturer les trames sur le port série.

Chaque nouvelle trame envoyée par le port série déclenchera un événement du type OnRxChar qui lui-même lancera la méthode acquis_serie.

Le timeout est fixé à 40.

Les trames GPS seront affichées dans un objet du type Tmemo:

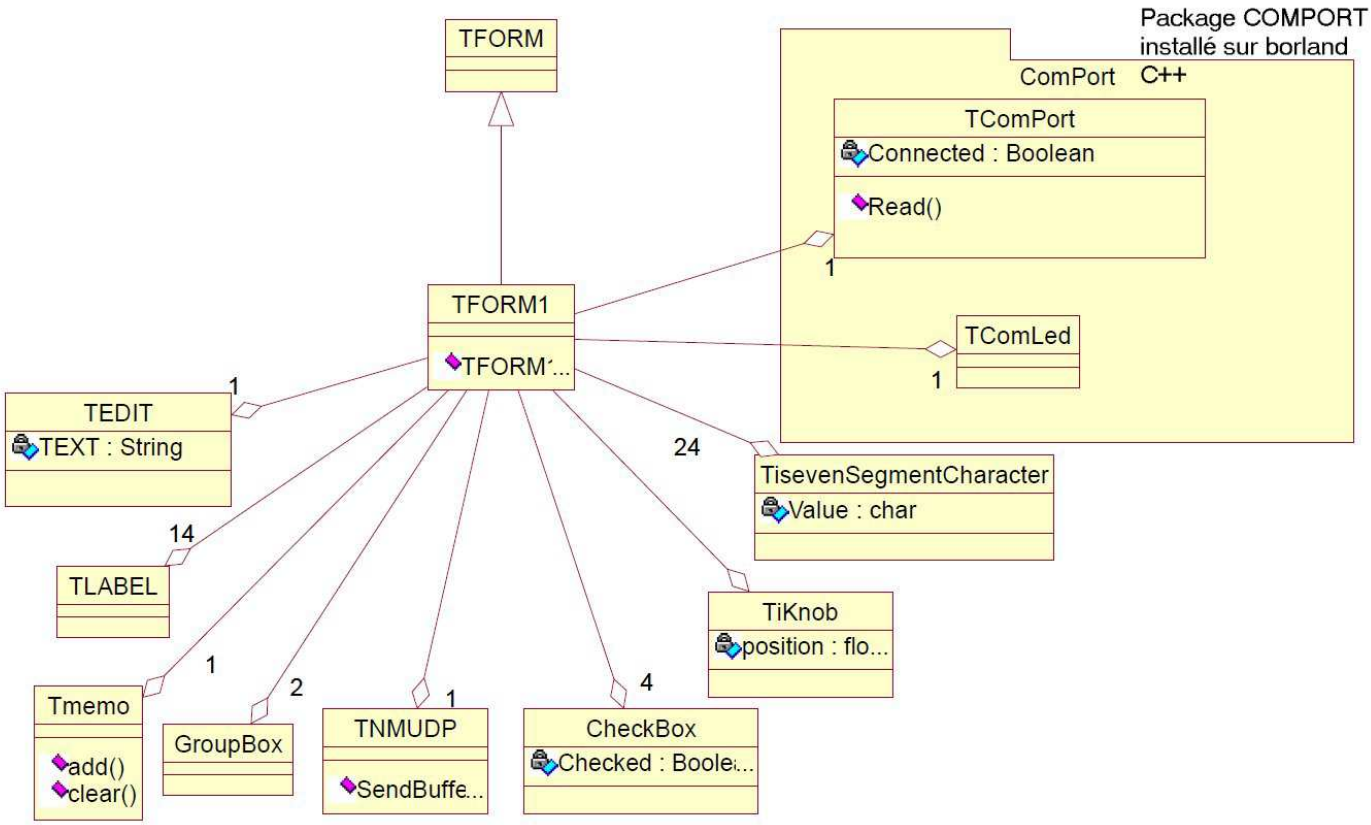
Pour chaque trame une nouvelle ligne sera créée.

Si la checkbox diffusion est cochée une trame NAV sera envoyée sur le réseau en diffusion générale. L'adresse de diffusion générale sera saisie grâce à un Tedit. Le mode de diffusion utilisé sera l'UDP sur le port 1000. Le format de la trame est le suivant :

char NAV[]={"4201.14N/00605.00E/000.0/08.0/**100.0**/0.00/00:09:26//"};

Les caractères en gras correspondent à l'immersion. Cette immersion sera réglée grâce à l'objet de classe TiKnob.

7. Diagramme de classes



8. Construction de l'IHM série

4.1) Réalisez l'interface graphique, en plaçant les différents objets sur la feuille form1.

Modifiez ci-nécessaire dans l'inspecteur d'objets, les propriétés ou les événements de chacun des objets.

4.2) La première partie consiste à récupérer les trames GPS sur le port série.

Branchez le récepteur GPS sur un ordinateur et en utilisant le programme hyperterminal, vérifiez le contenu de chaque trame

Créez la méthode acquis_serie, cette méthode est lancée à chaque événement OnRxChar.

On donne les déclarations de cette méthode

```
char NAV[]={"4201.14N/00605.00E/000.0/08.0/100.0/0.00/00:09:26//"};
//char NAV[]={'4','2','0','1','.','1','4','N','/','0','0','6','0',5','.','0','0','E','/
//4201.14N/00605.00E/000.0/08.0/016.0/0.00/00:09:26//
int i;
int  nbrvirg=0, indicetab;
char Buff[700];// tableau ou on va loger notre trame reçue
```

Les variables suivantes seront déclarées dans la classe TForm1 :

```
char Heure[10];// Heure GPS
char lattitude[12];// lattitude GPS
char longitude[12];// longitude GPS
char lattorientation[2];// orientation lattitude
char longorientation[2];// onrientation longitude
/* trame NAV    */
int immer=0;// valeur de l'immersion
unsigned char unite;// unité immersion caractère ASCII
unsigned char dizaine;// dizaine immersion caractère ASCII
unsigned char centaine;// centaine immersion caractère ASCII
```

a) L'immersion est capturée grâce au bouton. proposez les lignes de code qui permettent d'obtenir les caractères ASCII des centaines, dizaines, unités lorsqu'on règle le bouton. Modifiez ainsi le tableau Nav

b) Acquérir l'adresse IP du champ edit et affectez le composant de la classe TNMUDP avec cette adresse

c) Acquérir les trames grâce à la méthode read de la classe TComPort et affichez chaque trame sur la boite Memo

d) Acquérir les champs heure, latitude, longitude, orientation latitude, orientation longitude d'une trame du type GPGGA en se servant du nombre de virgule. L'Heure, la longitude, la latitude, l'orientation latitude, l'orientation longitude seront rangées respectivement dans les tableaux Heure, longitude, lattitude, lattorientation , longorientation.

e) Affectez chaque segment avec les caractères des tableaux Heure, longitude, lattitude, lattorientation , longorientation.

f) Modifiez le tableau NAV avec les caractères Heure, longitude, lattitude, lattorientation , longorientation.

g) Diffusez la trame NAV grâce à la méthode SendBuffer si la case à cocher diffusion est cochée

h) Testez votre programme

9) Gestion du temps réel

Lorsque les ressources du processeur sont limitées, il est difficile de cocher les cases, d'agir sur le bouton ou de saisir une adresse IP. En effet le programme est accaparé à recevoir les trames et ne répond plus aussi facilement aux sollicitations de l'utilisateur.
On va donc créer un thread qui permettra de saisir la valeur de l'immersion et d'afficher les informations GPS sur l'IHM. Pour vous aider on pourra utiliser le cours sur les processus légers ou thread.

a) Créez une classe du type thread et un objet monthread de cette classe

b) Modifiez la méthode AssignCtrl de façon à mettre à jour les informations GPS sur l'IHM et de saisir la valeur de l'immersion.

c) Testez à nouveau votre programme, conlure.

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
- 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)**

Les composants de la bibliothèque VCL

1) La classe TEdit

TEdit encapsule l'objet Windows " zone d'édition ", c'est à dire une petite zone dans laquelle l'utilisateur peut taper un texte.



Text: (Propriété)

Pour affecter un texte à un TEdit, il suffit d'affecter sa propriété Text :

```
Edit1->Text="Texte" ;
```

Pour lire le texte d'un TEdit, il suffit de lire sa propriété Text :

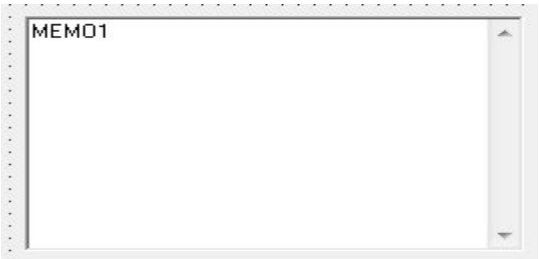
```
AnsiString s=Edit1->Text ;
```

Windows gère lui-même l'affichage du TEdit, le curseur texte, etc....

Il y a quelques événements associés au TEdit, comme OnChange qui indique que le texte contenu à l'intérieur à changé, ou OnKeyPress indiquant la frappe d'une touche.

2) La classe TMemo

Utilisez TMemo pour placer dans une fiche un contrôle de saisie multiligne Windows standard. Les boîtes de saisie multilignes permettent à l'utilisateur de saisir plusieurs lignes de texte. Ces composants sont appropriés pour la représentation de données volumineuses.



Principales méthodes, propriétés et événements:

Clear: (Méthode)

Efface le contenu du composant

Application:

Memo1->Clear() ;

Text: (Propriété)

Permet de connaitre le contenu (sous forme de TString) du composant. Si le contenu est multilignes, les lignes sont séparées les unes des autres par un caractère '\r' suivi du caractère '\n'

Application:

Memo1->Text = "Ligne 1 \r\n Ligne 2 \r\nTroisième ligne" ;

Lines: (Propriété)

Objet permettant d'atteindre le contenu du contrôle ligne par ligne. Il s'agit d'un objet de type TStrings. Toutes les méthodes et propriétés de TStrings peuvent donc être appliqués.

Application:

Memo1->Lines->Add("Ligne 4") ; // Ajoute une ligne contenant "Ligne 4"

Remarque concernant la méthode Add:

L'argument passé à cette méthode ne doit contenir ni '\n' ni '\r'. Sinon, cela provoque une erreur lors de l'éxécution

unsigned char tab[10] ="ici";
MEMO1->Lines->Add("Bonjour" +Buff+"fin");
On affichera sur une ligne Bonjouricifin

ScrollBars: (Propriété)

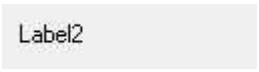
Détermine s'il faut ajouter des barres de défilement au contrôle mémo.
Les valeurs possibles sont **ssNone, ssHorizontal, ssVertical, ssBoth**

Application:

Memo1->ScrollBars=ssVertical ;

3) *La classe TLabel*

Contrôle Affichant un texte



Caption: (Propriété) : Ansistring

Permet de définir le texte qui va être affiché sur le label.

Application:

Label1->Caption = "Nom de la personne" ;

4) La classe TButton

Utilisez TButton pour placer un bouton poussoir standard dans une fiche. TButton introduit plusieurs propriétés permettant de contrôler son comportement dans la définition d'une boite de dialogue. Les utilisateurs choisissent des contrôles bouton pour effectuer une action.



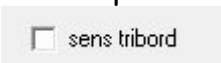
Les utilisateurs cliquent sur les contrôles bouton pour initier des actions. Vous pouvez associer une action à un composant TButton en associant une méthode à un événement OnClick correspondant (voir l'inspecteur d'objet).

OnClick: (événements)

L'événement Onclick se produit quand l'utilisateur clique sur le contrôle (ici le bouton). Utilisez le gestionnaire d'événement OnClick pour répondre quand l'utilisateur clique sur le contrôle.

5) La classe TCheckBox

TCheckBox représente une case à cocher qui peut prendre deux états : activé ou désactivé. Un composant TCheckBox propose une option à l'utilisateur. L'utilisateur peut activer la case à cocher pour sélectionner l'option, ou supprimer la case à cocher pour la désélectionner.



Checked: (Propriété)

Permet de savoir si la case est sélectionnée.

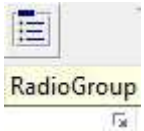
Application:

```
CheckBox1->Checked==true // test si la case CheckBox1 est cochée
```

6) La Classe TRadioButton

TRadioButton représente une case ou un ensemble de case à cocher qui peuvent prendre deux états : activé ou désactivé.

Contrairement à la classe TCheckBox, les cases sont toutes dépendantes entre elles et une seule case est cochée à la fois. Le fait de cocher une case décoche toutes les autres.



Cliquez sur Items pour définir les différents items à cocher



La boite de dialogue suivante s'ouvre :

Chaque ligne correspond à un item



Application:

RadioButton1->ItemIndex==0 // test si la case N°1 est cochée

7) La classe TNMUDP

Le composant TNMUDP est utilisé pour mettre en oeuvre le protocole UDP (User Datagram Protocol) pour l'envoi de paquets datagrammes sur Internet ou un intranet.



ReadBuffer: (méthode)

La méthode ReadBuffer lit des données du port UDP et les stocke dans un tampon.

int __fastcall ReadBuffer(char * buff, int lengthmin, int lengh);

Paramètres

Le paramètre Buff est le tampon dans lequel les données seront stockées.

Le paramètre length spécifie la quantité de données à lire.

Remarque
Le paramètre Buff doit être assez grand pour contenir les données arrivantes.
Si le tampon est trop petit, il se produit une violation d'accès.
Ecrit les tampons de données de taille bufsize dans le socket.

OnDataReceived: (événement)

Description
L'événement OnDataReceived est lancé quand des données sont reçues de l'hôte distant.
La methode NMUDPRECEIVE est appelée par cet événement :
NMUDPRECEIVE(TComponent *Sender, int NumberBytes,AnsiString FromIP, int Port)

Le paramètre NumberBytes spécifie le nombre d'octets reçus.
Le paramètre FromIP spécifie l'adresse IP de l'ordinateur qui a envoyé les données.
La propriété Port spécifie sur quel port les données ont été envoyées.

SendBuffer: (méthode)

Cette méthode permet de transmettre des caractères en utilisant le mode UDP-IP

Void **SendBuffer (char * tab, int size, int size) ;**

Paramètres
Le paramètre **tab** est le tampon dans lequel les caractères seront envoyées.
Le paramètre **size** spécifie la quantité de données à envoyer.

8) La classe TClientSocket

Ajoutez un objet TClientSocket à une fiche ou un module de données pour transformer une application en client TCP/IP. TClientSocket spécifie la connexion souhaitée avec un serveur TCP/IP, gère la connexion quand elle est ouverte et termine la connexion quand l'application est achevée.



Adress: (propriété) :ansistring

La propriété Address est une chaîne contenant quatre valeurs numériques (octet) utilisant la notation par point standard Internet, par exemple :
ClientSocket1->Address= "192.9.200.203"

Active: (propriété) :booléen

Avant d'utiliser ou de modifier la connexion de socket, consultez la propriété Active pour déterminer si la connexion est ouverte et prête. Pour les sockets client, l'affectation d'Active ouvre ou ferme une connexion de socket avec une autre machine. Pour les sockets serveur,

l'affectation d'Active ouvre ou ferme une connexion d'écoute qui rend le socket attentif aux demandes des clients.

```
ClientSocket1->Active = true ; // connexion établie
```

SendBuff: (méthode)

Ecrit les tampons de données de taille bufsize dans le socket.

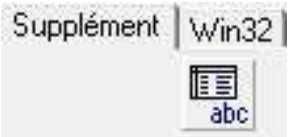
```
int __fastcall SendBuf(void *Buf, int BufSize);
```

SendBuf envoie les tampons de données de taille bufsize dans le socket. Utilisez buf pour affecter une variable au tampon, et bufsize pour définir la taille du tampon à envoyer.

```
unsigned char tab[10] = {0};
ClientSocket1->Socket->SendBuf(tab,sizeof(tab));
```

9) La classe TStringGrid

Le composant TStringGrid permet de presenter un tableau sur une interface homme machine.



Cells [ ] [ ]: (propriété) :ansistring

Pour lire ou écrire dans une case du tableau

```
StringGrid1->Cells[0][0] = "Lot";
```

Les composant VCL du package TCOMPORT

1) Présentation des composants ComPort

Le paquet de conception va installer une nouvelle palette de composants comportant :
- **TComPort** : composant d'accès au port série lui-même.
- **TComDataPacket** : Composant de gestion de trames reçues.
- **TComComboBox** : Composant visuel d'aide à la configuration du TComport
- **TComRadioGroup** : Composant visuel d'aide à la configuration du TComport
- **TComLed** : Composant permettant de visualiser l'activité du composant ComPort associé
- **TComTerminal** : Composant de visualisation des trames de la liaison série

Nous allons maintenant détailler l'utilisation de quelques un de ces composants.

2) TComPort

C'est le composant principal de ce paquet. C'est lui qui va permettre de configurer et de gérer le port série.
Ce composant comporte toutes les propriétés de configuration de la liaison série. Je ne vais pas toutes les présenter dans le détail, mais seulement les plus importantes et surtout celles que l'on modifie le plus souvent dans la pratique.

- Propriété **Port**

Cette propriété définit le port à utiliser sur le PC. Dans la liste déroulante seuls les ports existant physiquement apparaissent.
Notez que ComPort est basé sur les APIs Windows, donc il supporte parfaitement les drivers passerelle, style USB--RS232.

- Propriété **BaudRate**

Définit la vitesse de la liaison série en Bauds.

- Propriété **DataBits**

Définit le nombre de bits de chaque caractère émis/reçu sur la liaison série.

- Propriété **Parity**

Définit la type de bit de parité de chaque caractère. Il est possible ici de définir si le contrôle de parité doit être effectué en lecture, et dans le cas où il est effectué de remplacer dans la trame reçue les caractères ayant un défaut de parité par un caractère spécifique.
Pour les essais, je vous conseille de ne pas activer le contrôle de parité, il n'est pas indispensable au bon fonctionnement et peut facilement être ajouté par la suite si vous voulez renforcer le contrôle.
De toute manière, dans le cas d'envoi/réception de trames il est préférable de gérer un checksum plus évolué de style CRC.

- Propriété **StopsBits**

Nombre de bits de stops de chaque caractère.

- Propriété **FlowControl**

Cet ensemble de propriétés permet de définir la gestion du flux de caractères sur la liaison série.
Par expérience et dans la mesure du possible il est préférable de ne pas utiliser le contrôle de flux, car cela ne fait que compliquer le gestion et le cablâge de la liaison. Il y a cependant des exceptions, comme la gestion d'un modem ou la gestion d'un convertisseur RS232-RS485 multipoints.
Dans les premiers exemples donnés dans ce cours, aucun contrôle de flux ne sera utilisé.

- Propriété **Timeouts**

Cet ensemble de propriétés permet de définir les différents timeouts en émission et en réception.
Leur utilisation dépend des applications.

- Propriété **Events**

Permet de définir quels événements liés à l'activité de la liaison série sont utilisés.
Par défaut ils sont tous activés, mais vous pouvez supprimer tous ceux qui ne vous sont pas utiles afin de ne pas surcharger le dialogue avec le TComport.

- Propriété **EventChar**

Cette propriété permet de définir le caractère déclenchant **OnRxChar** quand il est reçu. Il faut bien sur que l'événement soit activé dans la propriété **Events**.

- Propriété **Connected**

Cette propriété permet d'ouvrir/fermer la liaison série et aussi d'en vérifier son état.

3) TComLed

Composant permettant de réaliser une animation visuelle d'activité de la liaison série.
La liste des propriétés est la suivante :

- Propriété **ComPort**

Permet de choisir à quel composant TComPort celui-ci doit être associé.
Cette propriété doit impérativement être définie.

- Propriété **King**

Permet de définir le type de dessin du composant. Dans le cas de **lkCustom**, vous devez donner des images pour les états ON et OFF dans les propriétés **GlyphON** et **GlyphOFF**.

- Propriété **LedSignal**

Permet d'associer le composant à un état particulier de la liaison série. Etat de la ligne de transmission ou de réception …..

- Propriété **State**

Permet de définir, mais surtout de connaître l'état ON/OFF du composant. Quand la liaison est active, cette propriété est en lecture seule.

4) Quelques exemples

Ouverture du port

On définit préalablement, le port COM1, 2, 3 ou 4 dans l'inspecteur d'objets, (propriété Port).

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
ComPort1->Connected=true; //Ouverture du port (prêt à //communiquer)
}
//---------------------------------------------------------
```

Le port et maintenant ouvert et prêt à communiquer avec un périphérique externe.

Fermeture du port

```
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
ComPort1->Connected=false; //Fermeture du port (arrêt des //communications)
}
//---------------------------------------------------------
```

Ecrire une chaîne de caractères sur TxD

Vous serez peut être amené à envoyer des caractères ASCII vers le correspondant.
Voilà un petit exemple de la manière à procéder pour faire cela.

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
AnsiString Phrase;
Phrase = Edit1->Text;
ComPort1->WriteStr(Phrase); //Écrit toute la chaîne "Phrase" //sur le port série
}
```

Ecrire une valeur sur TxD

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
unsigned char tableau[7] = {'V', 'i', 'n', 'c', 'e', 'n', 't'};
ComPort1->Write(tableau, 7); //Écrit 7 octets de "tableau" sur //le port série
}
```

```
//---------------------------------------------------------
```

Lecture d'une chaîne de caractères sur RxD

Après avoir envoyé une chaîne de caractères sur la ligne TxD, il serait bien de savoir comment la recevoir.
Écrivons dans l'événement OnRxChar:

```
//---------------------------------------------------
void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
AnsiString Phrase2;
ComPort1->ReadStr(Phrase2, Count);
//Lit les "Count" octet(s) présent(s) dans le buffer d'entrée //et le(s)place dans
Phrase2
}
//---------------------------------------------------
```

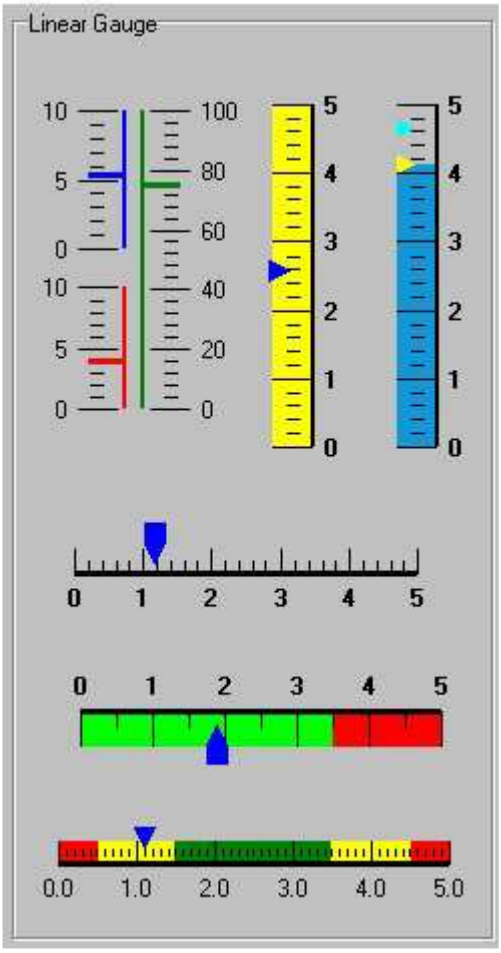
La phrase reçue est stockée dans le buffer d'entrée du port, dont la taille est paramétrable dans l'inspecteur d'objets, propriété Buffer / Input et OutputSize.
Dans cet exemple, nous remarquons bien que la lecture se fait caractère par caractère, ReadStr vient effacer le caractère dans le buffer, puis il passe au suivant et refait de même.

```
void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
Char tab[70];
ComPort1->Read(tab, 70);  //Lit les 70 octet(s) présent(s) dans le buffer d'entrée
//et le(s)place dans Phrase2
}
```

Les composant VCL du package IOCOMP

1) La classe TiLinearGauge

TiLinearGauge est une gauge graphique avec une représentation linéaire des données. Ce composant supporte jusqu'à 5 sections et 4 styles de pointeur



Position: (Propriété)

Description

Déplace le pointeur à la position données

iComponent->Position = 65;

SectionCount: (Propriété)

Spécifie le nombre de sections de couleur

SectionColor1: (Propriété)

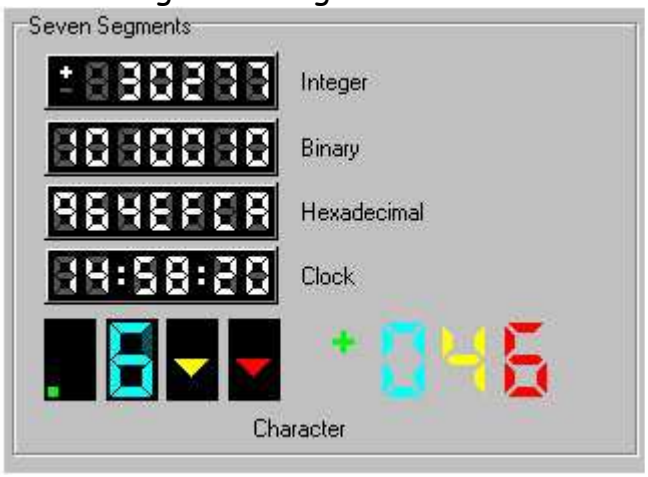
Spécifie la couleur de la section N°1

SectionEnd1: (Propriété)

Spécifie la valeur de fin pour la section N°1. La valeur sera comprise entre PositionMin et PositionMax.

2) La classe TiSevenSegmentinteger

TiSevenSegmentInteger est un afficheur 7 segments pour un affichage de valeur entière.



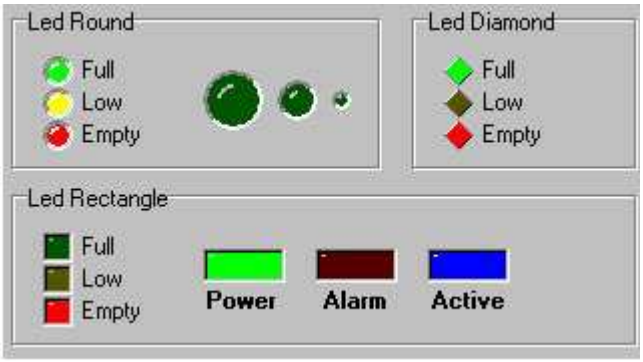
- **Value: (Propriété):entier**

Utilise la propriété Value pour changer la valeur du chiffre affichée

```
iComponent->Value = 254;
```

3) La classe TiLedRound

TiLedRound est une LED



- **Value: (Propriété):Boolean**

```
iComponent->Active = True; // La LED est allumée
```

4) La classe TiXYPLOT

Ce composant permet d'afficher un graphe du type XY



- **AddChannel(): (méthode):void**

Cette méthode permet de créer une nouvelle courbe .

iXYPlot1->AddChannel();

- **RemoveAllChannels(): (méthode):void**

Cette méthode permet d'effacer toutes les courbes.

- **Channel[nchannel]: (objet):void**

Cette méthode permet de sélectionner une courbe d'indice nchannel

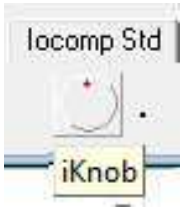
- **AddXY(double X,double Y): (méthode):void**

Cette méthode permet de tracer un point d'abscisse X et d'ordonnées Y

iXYPlot1->Channel[nchannel]->AddXY(3,4); // Le point d'abscisse 3 et d'ordonnées 4 est tracé

5) La classe TiKnob

Ce composant permet d'acquérir une consigne par l'intermédiaire d'un bouton rotatif



- **Position: (Propriété):double**

Grâce à ce membre on peut récupérer la valeur régler sur le bouton, ou modifier le réglage du bouton

iKnob1->Position;

Quelques fonctions utiles

Float StrToFloat (Ansistring chiffre)

Convertir un chiffre sous forme de chaine de caractère en un float

Ansistring FloatToStr (float chiffre)

Convertir un floatant en une chaine de caractère

Int StrToInt(Ansistring chiffre)

Convertir un chiffre sous forme de chaine de caractère en un int

La directive __fastcall

Utilisez le plus souvent possible la directive **__fastcall** dans la déclaration de vos méthodes. Elle signifie simplement que la transcription en code machine, au moment de la compilation, sera optimisée par l'utilisation des registres internes du microprocesseur.

Les classes Tobject et TComponent

Tobject est la classe père de tous les objets de Borland C++. pour que la compatibilité soit assurée, on fera passer un pointeur sur l'objet qui est à 'initiative des événements (onclick() par exemple).
TComponent est la classe père de tous les composants ou objets de la VCL de Borland C++. pour que la compatibilité soit assurée, on fera passer un pointeur sur un objet de la VCL lors de la construction de la feuille.

Le Mot-Clé published

Chaque composant du Builder (une classe en fait) contient un certain nombre de propriétés (données) qui sont visibles dans l'inspecteur d'objets, et qui peuvent être modifiés durant la phase de développement.

Dans la déclaration de la classe (fichier .h en général), chacune de ces donnée est précédée du mot clé __published. Les deux traits __ indiquent que ce mot-clé est propre au C++ builder et n'existe pas (encore) en C++ standard.

```
#include <qapplication.h>
#include <qpushbutton.h>
```

```
int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qwidget.h>
int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
bouton->show();
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qwidget.h>
int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->show();
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qcheckbox.h>
```

```
#include <qwidget.h>
//#include <qobject.h>

int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
checkbox->show();
bouton->show();
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qcheckbox.h>
#include <qwidget.h>
//#include <qobject.h>

int main( int argc, char **argv )
{
QApplication a( argc, argv );
QWidget fenetre;
fenetre.setFixedSize(640, 480);
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->move(70, 60);
checkbox->setGeometry(200,200,200,30);
fenetre.show();
return a.exec();
}
```

```
#include <qwidget.h>
//#include <qobject.h>
```

```
int main( int argc, char **argv )
{
QApplication a( argc, argv );
QString text1;
QWidget fenetre;
fenetre.setFixedSize(640, 480);
QLineEdit *vitesse=new QLineEdit("", &fenetre);
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->move(70, 60);
vitesse->move(100, 100);
vitesse->setText("bonjour");
text1=vitesse->text();
checkbox->setGeometry(200,200,200,30);
fenetre.show();
return a.exec();
}

#include <qapplication.h>
#include <qpushbutton.h>
#include <qlineedit.h>
#include <qcheckbox.h>
#include <qwidget.h>
#include <qspinbox.h>
//#include <qobject.h>

int main( int argc, char **argv )
{
int b;
QApplication a( argc, argv );
QString text1;
QWidget fenetre;
fenetre.setFixedSize(640, 480);
//QSpinBox *vit=new QSpinBox(&fenetre);
QLineEdit *vitesse=new QLineEdit("", &fenetre);
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->move(70, 60);
//vit->move(100, 100);
//b=vit->value();
vitesse->move(100, 100);
vitesse->setText("123");
text1=vitesse->text();
b=text1.toInt();
checkbox->setGeometry(200,200,200,30);
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qlineedit.h>
#include <qcheckbox.h>
#include <qwidget.h>
#include <qspinbox.h>
//#include <qobject.h>

int main( int argc, char **argv )
{
int b;
QApplication a( argc, argv );
QString text1;
QWidget fenetre;
fenetre.setFixedSize(640, 480);
//QSpinBox *vit=new QSpinBox(&fenetre);
QLineEdit *vitesse=new QLineEdit("", &fenetre);
QLineEdit *chiffre=new QLineEdit("", &fenetre);
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->move(70, 60);
//vit->move(100, 100);
//b=vit->value();
vitesse->move(100, 100);
vitesse->setText("123");
text1=vitesse->text();// Conversion
b=text1.toInt();// Conversion de string en int
text1=(QString::number(b,10));// conversion de int en string
chiffre->setText(text1);
chiffre->move(150, 150);
checkbox->setGeometry(200,200,200,30);
fenetre.show();
return a.exec();
}
```

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qlineedit.h>
```

```
#include <qcheckbox.h>
#include <qwidget.h>
#include <qspinbox.h>
#include <qlcdnumber.h>
//#include <qobject.h>

int main( int argc, char **argv )
{
int b;
QApplication a( argc, argv );
QString text1;
QWidget fenetre;
QLCDNumber *LCD= new QLCDNumber(5,&fenetre);
fenetre.setFixedSize(640, 480);
//QSpinBox *vit=new QSpinBox(&fenetre);
QLineEdit *vitesse=new QLineEdit("", &fenetre);
QLineEdit *chiffre=new QLineEdit("", &fenetre);
QCheckBox *checkbox = new QCheckBox("J'aime les frites", &fenetre);
QPushButton *bouton =new QPushButton("&Quitter", &fenetre);
QObject::connect(bouton, SIGNAL(clicked()),&a, SLOT(quit()));
bouton->move(70, 60);
//vit->move(100, 100);
//b=vit->value();
vitesse->move(100, 100);
vitesse->setText("123");
text1=vitesse->text();// Conversion
b=text1.toInt();// Conversion de string en int
text1=(QString::number(b,10));// conversion de int en string
//chiffre->setText(text1);
//chiffre->move(150, 150);
LCD->move(300, 300);
LCD->display(b);
checkbox->setGeometry(200,200,200,30);
fenetre.show();
return a.exec();
}
```