

Créez une interface avec GTK+ !

Par guimers8 ,
Im@GinE ,
JRG Soft ,
Petrus6
et Picxime



www.openclassrooms.com

Sommaire

Sommaire	2
Partager	1
Créez une interface avec GTK+ !	3
Partie 1 : Les bases de GTK	3
Avant de se lancer.....	4
L'Histoire de GTK	4
Les avantages	4
Possibilités de GTK	5
GTK ça ressemble à quoi ?	5
Et qu'est-ce qu'on peut faire avec ?	6
Installer GTK+ sous Windows	6
Télécharger le Pack GTK+ & le Runtime	7
Installation	7
Le Runtime GTK+	8
Le Pack GTK+	8
Configuration du Logiciel & d'un Projet GTK+	9
Configuration du Logiciel	9
Configuration d'un Projet	12
Astuces	17
Installer GTK+ sous Linux	18
Télécharger et installer les fichiers	19
Mode Console	19
Mode Graphique	20
Compilation en ligne de commande	20
Compilation sous Code::Blocks	21
Configuration du logiciel	21
Test de la configuration	25
Installer GTK+ sous MacOS X	26
Fink, X11 ?	27
Installation	27
XCode	27
X11	27
Fink	28
Compilation	28
Configuration du projet	28
Compilation et exécution	29
Notions de base et fenêtres	31
Widgets et héritage	31
Les Widgets	31
L'héritage	31
Code de base et création d'une fenêtre	32
Code de base GTK	32
Création d'une fenêtre	33
Personnaliser notre fenêtre	35
Le titre	36
La taille (par défaut)	36
La taille (actuelle)	37
Positionnement	37
L'icône	37
Iconifier	38
Maximiser	38
Bouton "fermer"	38
Les bordures	39
Exemple	39
Le texte avec les labels	39
Les GtkLabels	40
Créer un label	40
Récupérer et modifier un label	41
Les Accents	41
Changer le style d'un label	42
L'alignement	42
Le formatage du texte avec les balises	42
Exemple	44
Gestion des événements : les signaux	46
Comment ça marche ?	46
Exercices	48
Partie 2 : Annexes	49
Trucs et astuces divers	50
Quelques Macros.....	50
Retour à la console	50
Problèmes d'encodage UTF-8	50
Texte accentué	51

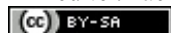


Créez une interface avec GTK+ !



Mise à jour : 24/09/2010

Difficulté : Facile



Ce tutoriel n'est plus mis à jour

Le BIG tuto GTK n'est plus maintenu.

Avec l'équipe nous allons voir comment le réorganiser et enlever ce qui ne va plus pour vous éviter d'éventuels problèmes.

Quant à le continuer, pour le moment ce n'est pas envisageable. Peut être qu'un membre souhaiterait le faire ?



Apprendre à programmer, c'est bien !
Mais ensuite, qu'en faire ?

La bibliothèque GTK+, écrite en C permet de créer une interface graphique pour vos applications.

Vous pouvez créer fenêtres, boutons, menus, barres d'outils, barres de progression, etc... et il est même possible d'imprimer, d'appliquer un thème, et bien d'autres choses encore ! 😊

Nous sommes trois membres du site a nous être rassemblés pour tout vous apprendre sur cette bibliothèque :

[Picxime](#), [Guimers8](#) et [Im@GinE](#).

Dans ce cours complet, nous vous apprendrons progressivement comment créer et utiliser les différents objets de la bibliothèque et nous ferons quelques TPs qui devraient vous intéresser !

Si l'aventure GTK+ vous tente, pourquoi ne pas essayer, c'est facile et très intéressant ! 😊

Partie 1 : Les bases de GTK

Parce qu'il faut bien commencer par quelque part, cette partie posera les bases et quelques notions essentielles pour bien démarrer. 😊

📁 Avant de se lancer...

Bienvenue à tous dans ce premier chapitre ! 😊

Si vous êtes là, c'est sûrement que vous souhaitez découvrir GTK+, ses possibilités, son apparence, etc... C'est donc ce que je vais tenter de vous présenter, vous pourrez ainsi voir si vous voulez apprendre à vous en servir. 😊

L'Histoire de GTK

GTK est née d'un projet bien particulier, celui de créer une librairie graphique portable pour faire un logiciel de retouche d'images. Cette idée est venue à l'esprit de **Peter Mattis, Spencer Kimball & Josh MacDonald**, GTK venait de naître !



Le logo de GTK

❓ Oui mais alors, pourquoi GTK ?
Après tout, que signifie ce nom bizarre ???

Ce nom signifie *The GIMP ToolKit* !

The GIMP étant bien entendu le nom de leur logiciel de retouche et *ToolKit* signifiant Kit d'outils (en gros 😊).

À la base, la bibliothèque GTK était faite pour être utilisée par The GIMP, mais maintenant les choses ont légèrement changé. Certes, GTK est toujours utilisée par The GIMP, mais de plus, GTK est maintenant utilisée par de nombreux projets (plus ou moins gros), comme par exemple Gnome.

Les avantages

❓ Quels sont les avantages de GTK ?
C'est vrai ça ? Pourquoi on prendrait GTK et pas une autre GUI ?

GTK possède plusieurs avantages (je ne dis pas que les autres GUI ne les ont pas, mais GTK les a) :

- **Licence GNU LGPL** : GTK est libre, gratuite & modifiable à souhait (utilisable pour faire des logiciels payants sans acheter de licence).
- **Multi-Plateforme** : GTK existe sur de nombreuses plateformes comme Linux, Windows, Mac OSX, BSD et BeOS. Vive le portable ! 😊
- **Multi-Language** : On peut programmer avec GTK avec des très nombreux langages comme le C, C++, Ada, C#, Java, Python, Perl, PHP, Pascal, Fortran, Eiffel, etc... 😊

À l'appui, une citation de notre grand ami :

Citation : M@teo21

C'est une librairie de fenêtres multiplateforme. Elle a été créée au départ pour le logiciel de dessin The Gimp, puis elle a été étendue et améliorée pour être utilisable par d'autres programmes. Contrairement à la SDL qui ne permet pas de créer des boutons et de menus (enfin c'est possible mais il faut les simuler c'est un peu délicat) et qui est plutôt adaptée pour les jeux, GTK+ vous propose tout ça. C'est une librairie sous licence LGPL aussi, donc vous êtes libres de distribuer vos programmes comme vous le voulez.

Pour info, sachiez-vous que...

- GTK a été portée en plus de **32 langages** différents ?
- Vous pouvez télécharger plus de **200 thèmes** différents pour vos applications, et même créer le votre ?
- Derrière le nom GTK se cache une acronymie imbriquée relativement complexe :
GTK : the GIMP ToolKit.
GIMP : GNU Image Manipulation Program.
GNU : GNU's Not UNIX.
UNIX : Uniplexed Information and Computing Service.
 Soit GTK : The GNU's Not Uniplexed Information and Computing Service Image Manipulation Program Toolkit...
 (Un sacré morceau, pas vrai !? 🤪)

Sources et sites webs

- **Site Officiel** : www.gtk.org
- **GNU** : www.gnu.org
- **LGPL** : fr.wikipedia.org/LGPL
- **Projet GNOME** : www.gnomefr.org

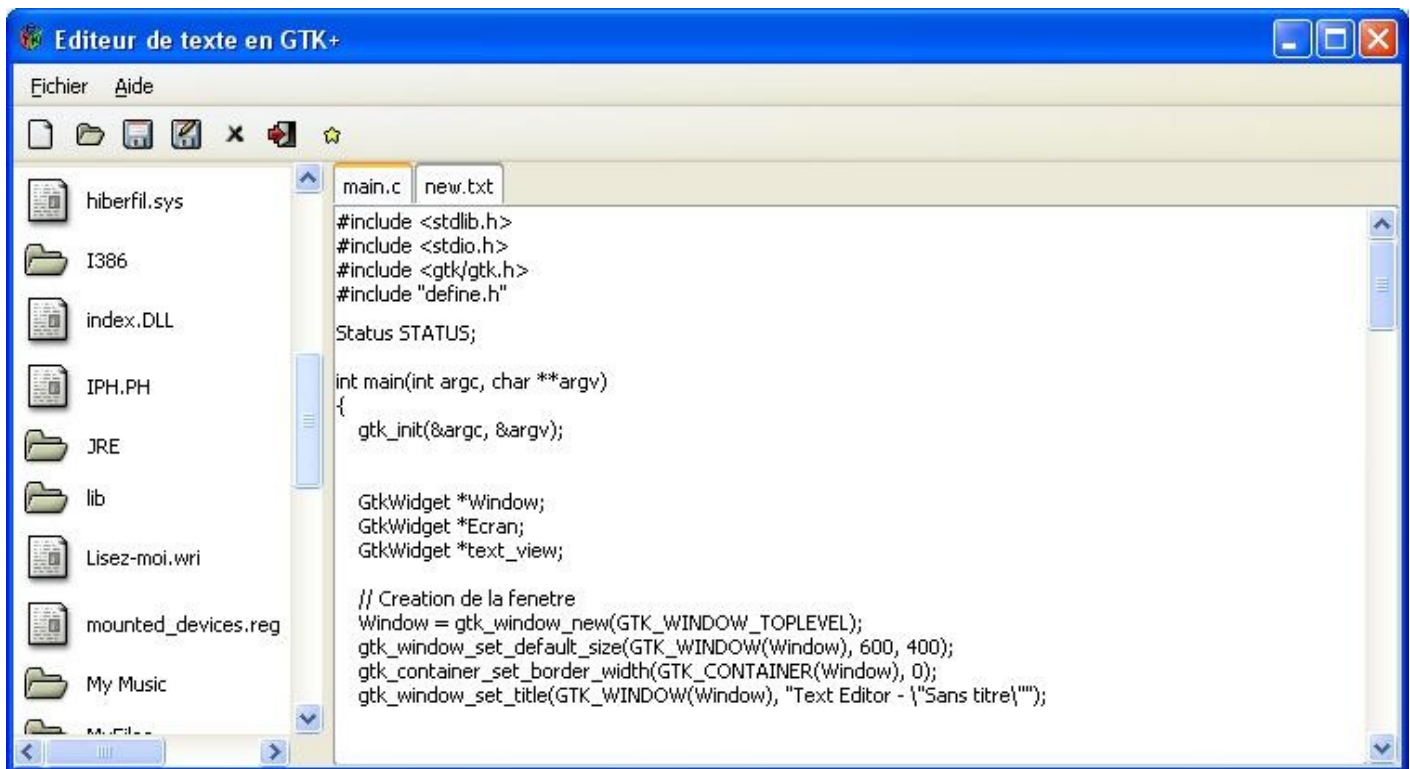
Possibilités de GTK

GTK ça ressemble à quoi ?

GTK est donc (avec les bibliothèques qui l'accompagnent) une bibliothèque graphique qui permet de créer des programmes de type GUI (programme à interface graphique) comprenant fenêtres, boutons, zones de texte, etc... 😊

Cependant, les applications GTK ont une apparence bien particulière, beaucoup de gens d'incultes appellent (à tort) les applications GTK, *fenêtres Linux* ! 🤪

Vous allez voir pourquoi :



Un éditeur de texte en GTK sous XP; ça change pour les Windoziens, mais quand on y pense, c'est beau quand même ! 😊

J'aurais pu prendre un exemple plus flagrant, c'est vrai, mais on voit quand même que ce n'est pas de l'API Win32 ou du Cocoa Mac OSX !

Les linuxiens sous *Gnome* ne seront absolument pas dépaysés ! 🤪

Mais si cela ne vous plaît pas (on ne sait jamais), vous pourrez toujours utiliser les thèmes GTK pour modifier l'apparence de vos applications ! 😊 (On verra comment faire...)

Si vous voulez voir d'autres images, cherchez **The GIMP**, **Inkscape**, ou encore **Gnome** sur *Google Images*. 😊

Et qu'est-ce qu'on peut faire avec ?

GTK permet (et c'est son but principal) de créer une interface graphique pour faire interagir l'utilisateur autrement qu'à grands coups de *scanf*, vous pourrez donc ajouter/enlever à volonté différents éléments dans vos fenêtres !

Voici une liste non-exhaustive des éléments que vous pouvez créer :

- Fenêtres
- Boîtes de dialogues
- Sélections de couleur, Polices ou de Fichiers
- Impressions de documents
- Menus, Barres d'outils, Barres de Statut
- Affichage de textes et de liens
- Images
- Frames, cadres
- Boîtes et tableaux
- Boutons, cases à cocher et zones d'options
- Zones de textes (simple & multi-lignes)
- Barres de progression
- Onglets
- Listes et arbres
- Zones de dessin
- Création et utilisation de thèmes !
- Intégration d'un contexte OpenGL ou d'une surface SDL

... Et sûrement encore un certain nombre d'éléments, mais bon, ça fait déjà un paquet, non ? 😊

Elle vous fait envie cette petite liste, ça se voit ! 😊

Après cette brève présentation de la bibliothèque, vous savez maintenant ce qui vous attend.

Si vous êtes tenté, passez au deuxième chapitre et installez GTK !!! 😊

Rassurez-vous, GTK est très facile à apprendre, et puis si vous n'êtes pas sûrs, essayez, vous verrez bien que vous auriez eu tort de passer à côté ! 😊

By [Guimers8](#)

Installer GTK+ sous Windows

Ce tutoriel n'est plus mis à jour, nous vous recommandons d'aller télécharger GTK sur le site officiel.

- [Windows 32bits](#)
- [Windows 64bits](#)



Le BIG tuto GTK n'est plus maintenu.

Avec l'équipe nous allons voir comment le réorganiser et enlever ce qui ne va plus pour vous éviter d'éventuels problèmes.

Quant à le continuer, pour le moment ce n'est pas envisageable. Peut être qu'un membre souhaiterait le faire ?

Bonjour à tous,

Dans ce chapitre, je vais vous expliquer comment installer la bibliothèque GTK+ (version **2.10.12**). Si certains ont déjà essayé de l'installer, ils ont dû se rendre compte que c'était vraiment le bazar, je confirme... 😊

Pour que vous n'ayez pas à télécharger plein de fichiers, je les ai regroupés dans un programme d'Installation ! 🎩

Donc dites-vous que vous avez la chance d'avoir devant vous, un tuto qui vous explique comment installer cette bibliothèque pour Dev-C++, Visual C++ (fonctionne avec la version Express) et Code::Blocks 😊.

Télécharger le Pack GTK+ & le Runtime

Pour commencer, téléchargez le Pack GTK+ que j'ai réalisé.

[Pack GTK+ 2.10 \(22,8Mo\)](#)

Ce Pack contient :

- GTK+ 2.10.12
- GLib 2.13.1
- ATK 1.19.1

- Cairo 1.2.6
- Pango 1.16.4
- Gettext 0.14.5
- Libiconv 1.9.1
- Libpng 1.2.8
- Tiff 3.7.4
- Zlib 1.2.3 dll

Ensuite, téléchargez le [Runtime GTK+ \(8.1Mo\)](#).



Le [Runtime](#) est un programme qui va installer des DLL, etc... dans les ressources de Windows pour que vos applications GTK+ puissent fonctionner ! 😊

Voilà, nous sommes maintenant prêts pour Installer GTK+ ! 🧑

Installation

Nous entrons à présent dans le vif du sujet : l'installation ! 😊



Si vous avez déjà essayé d'installer la bibliothèque GTK+, assurez-vous d'avoir désinstallé et supprimé tous les fichiers que vous avez ajoutés. Et si vous avez modifié les options de votre logiciel, je vous conseille même de le réinstaller pour être sûr qu'il n'y ait pas de problème(s) mais si vous avez déjà installé GTK+ avec mon tutorial dans une ancienne version du Pack et que vous voulez seulement mettre à jour votre version de GTK+, désinstallez le Pack avant d'installer le nouveau et vous aurez juste à modifier les chemins des dossiers dans les options de votre logiciel pour que tout soit à jour ! 😊

Le Runtime GTK+

Lancez l'installation du Runtime GTK+ et **installez-le dans les ressources de Windows**.

Donc si vous avez Windows XP ou Vista, dans **C:\WINDOWS\System32**

ou si vous avez Windows 2000 ou une version antérieure, dans **C:\WINNT\System32**

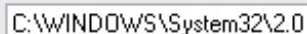
Le Runtime copie seulement des fichiers, il n'y a donc aucun risque de l'installer dans les ressources de Windows.



Il est **important** d'installer le Runtime dans les ressources de Windows !

Environ 27 Mo sont nécessaires !

Il est possible que lorsque vous aurez entré le chemin d'installation, le programme ajoute **2.0** :



C:\WINDOWS\System32\2.0

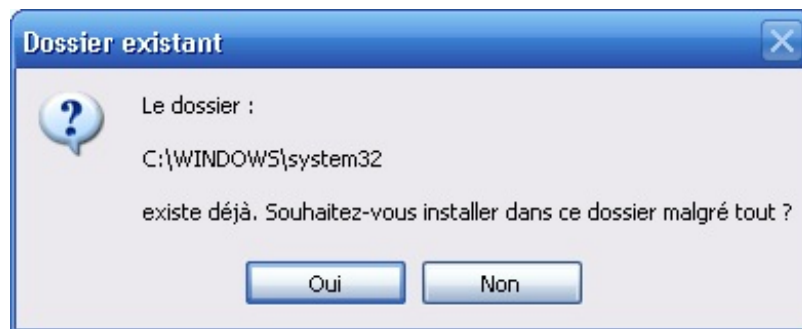
Supprimez-le pour que le Runtime ne soit pas installé dans un dossier nommé **2.0** :



C:\WINDOWS\System32



Ensuite, lorsque vous cliquez sur *Suivant*, le programme peut afficher ce message :



Cliquez sur *Oui*.

Le Pack GTK+

Comme je vous l'ai promis, vous n'aurez pas à télécharger plein de fichiers, à décompresser et à copier grâce au programme que j'ai réalisé ! 🤖

Lancez donc l'installation du Pack GTK+ que vous avez téléchargé :



Setup_Pack_G...

Choisissez *Français* (ou une autre langue, il ne doit pas y avoir que des francophones qui lisent ce tuto ! 🤖),

Cliquez sur *Suivant*, acceptez le *Contrat de License* et installez-le dans :

C:\Program Files\Pack GTK+



Il est **important** d'installer le Pack GTK+ à cet endroit ! 🤖

La lettre **C** du chemin du dossier correspond au disque dur local de votre PC. Si le vôtre n'a pas cette lettre, mettez en une autre; mais attention : pour tout le tuto vous devrez changer la lettre des chemins.



Environ 185 Mo sont nécessaires !

Maintenant que le Pack et le Runtime ont été installés, je vous conseille de redémarrer votre PC pour que Windows prenne bien en compte les deux installations.

Configuration du Logiciel & d'un Projet GTK+

Configuration du Logiciel

Maintenant que les fichiers ressources ont été copiés sur votre disque dur (pendant l'installation), il faut configurer votre logiciel pour qu'il puisse trouver les fichiers .h, .lib, etc... dont il a besoin lors de la compilation. D'abord, je vais vous donner les lignes à ajouter dans les options du compilateur et ensuite je vous explique comment faire 😊.



Chaque logiciel nécessite une installation différente, donc soyez bien attentifs ! 😊

Voici les lignes à ajouter dans les options du compilateur,

Répertoire du Dossier *Bin* (contenant les dll, etc...) :

C:\Program Files\Pack GTK+\bin

Répertoire du dossier *Lib* (contenant le fichiers .lib) :

C:\Program Files\Pack GTK+\lib

Répertoire des dossiers *Include* (contenant les fichiers .h et .c) :

C:\Program Files\Pack GTK+\include
C:\Program Files\Pack GTK+\include\atk
C:\Program Files\Pack GTK+\include\cairo
C:\Program Files\Pack GTK+\include\cairo\src
C:\Program Files\Pack GTK+\include\glib
C:\Program Files\Pack GTK+\include\glib\build
C:\Program Files\Pack GTK+\include\glib\glib
C:\Program Files\Pack GTK+\include\glib\gmodule
C:\Program Files\Pack GTK+\include\glib\gobject
C:\Program Files\Pack GTK+\include\glib\gthread
C:\Program Files\Pack GTK+\include\gtk+
C:\Program Files\Pack GTK+\include\gtk+\contrib
C:\Program Files\Pack GTK+\include\gtk+\gtk
C:\Program Files\Pack GTK+\include\gtk+\gdk
C:\Program Files\Pack GTK+\include\gtk+\gdk-pixbuf
C:\Program Files\Pack GTK+\include\gtk+\modules
C:\Program Files\Pack GTK+\include\libpng
C:\Program Files\Pack GTK+\include\libpng\contrib
C:\Program Files\Pack GTK+\include\pango
C:\Program Files\Pack GTK+\include\pango\modules
C:\Program Files\Pack GTK+\include\pango\pango
C:\Program Files\Pack GTK+\include\tiff
C:\Program Files\Pack GTK+\include\tiff\libtiff
C:\Program Files\Pack GTK+\lib\glib\include
C:\Program Files\Pack GTK+\lib\gtk+\include



Pour Dev-C++

Dans le menu de Dev-C++, allez dans *Outils / Options du Compilateur*, puis dans l'onglet *Répertoires*. Ensuite, dans les sous-onglets :



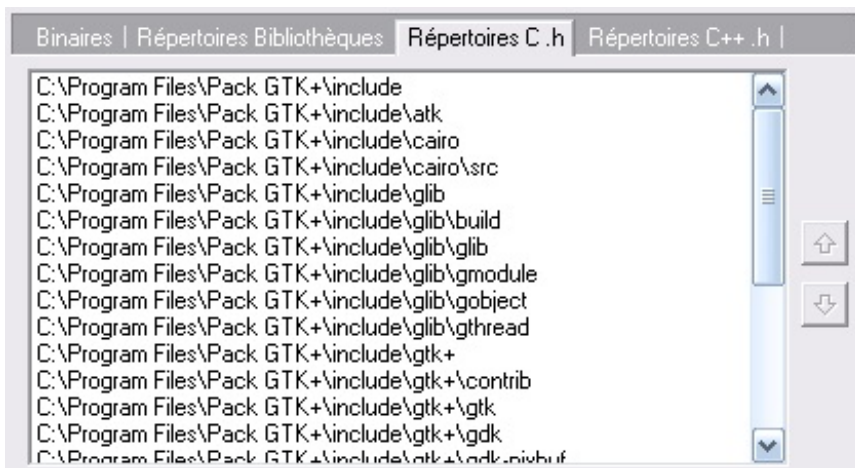
Cliquez sur *Binaires* et ajoutez une ligne dans laquelle vous entrez le chemin du répertoire du dossier *Bin* (reportez-vous à ce que j'ai écrit plus haut).



Dans l'onglet *Répertoires Bibliothèques*, ajoutez une ligne dans laquelle vous entrez le chemin du répertoire du dossier *Lib*.



Et pour finir, dans l'onglet *Répertoires C .h*, ajoutez des lignes dans lesquelles vous entrez les chemins des répertoires des dossiers *Include*.



(Faut les faire une par une ! 😊)
Puis cliquez sur *OK*.



Dans le menu de Visual C++, cliquez sur *Outils / Options...*

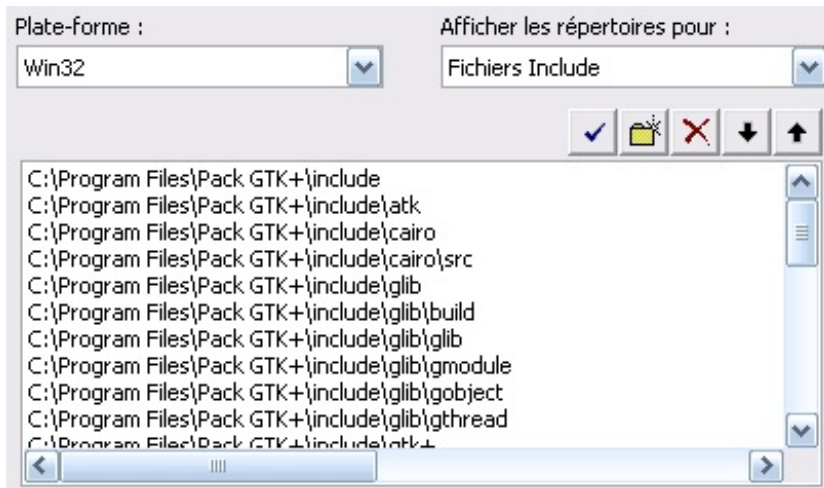
A gauche, dans la sélection, allez dans *Projets et solutions / Répertoires de VC++* puis à droite de la fenêtre, sélectionnez *Afficher les répertoires pour Fichiers exécutables* et ajoutez une ligne dans laquelle vous entrez le chemin du répertoire du dossier *Bin* (reportez-vous à ce que j'ai écrit plus haut).



Sélectionnez ensuite *Afficher les répertoires pour Fichiers bibliothèques* et ajoutez une ligne dans laquelle vous entrez le chemin du répertoire du dossier *Lib*.



Et pour finir, sélectionnez *Afficher les répertoires pour Fichiers Include* et ajoutez des lignes dans lesquelles vous entrerez les chemins des répertoires des dossiers *Include*.

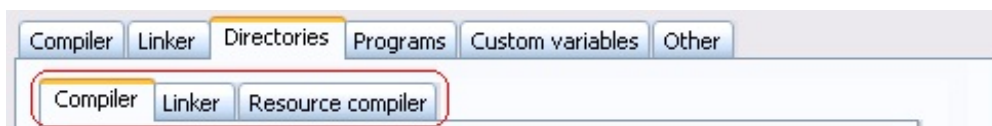


Cliquez sur OK pour appliquer les modifications.

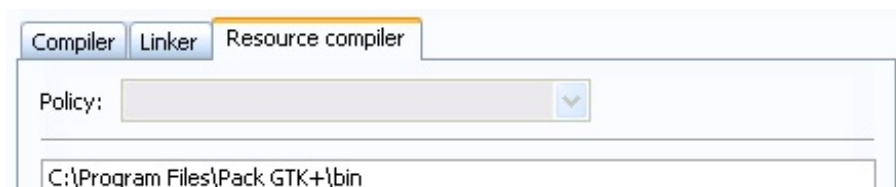


Pour Code::Blocks

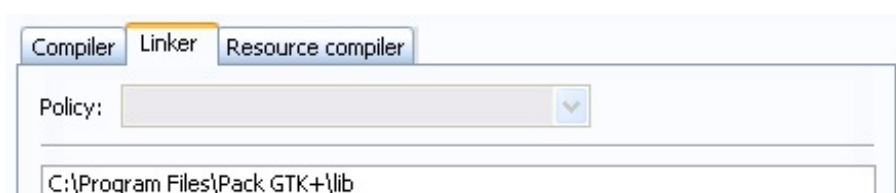
Dans le menu de Code::Blocks, cliquez sur Settings / Compiler, puis allez dans l'onglet Directories. Ensuite, dans les sous-onglets :



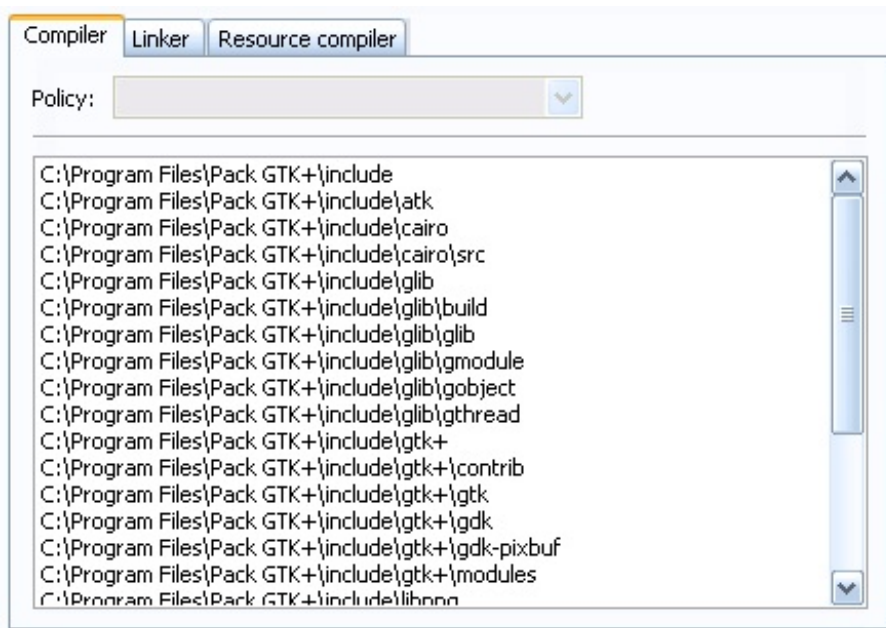
Cliquez sur *Resource compiler* et ajoutez une ligne dans laquelle vous entrerez le chemin du répertoire du dossier *Bin* (reportez-vous à ce que j'ai écrit plus haut).



Dans l'onglet *Linker*, ajoutez une ligne dans laquelle vous entrerez le chemin du répertoire du dossier *Lib*.



Et pour finir, dans l'onglet *Compiler*, ajoutez des lignes dans lesquelles vous entrerez les chemins des répertoires des dossiers *Include*.



Puis cliquez sur *OK*.

Voilà, vous avez terminé de configurer votre logiciel ! 😊 Mais c'est pas fini ! 😊

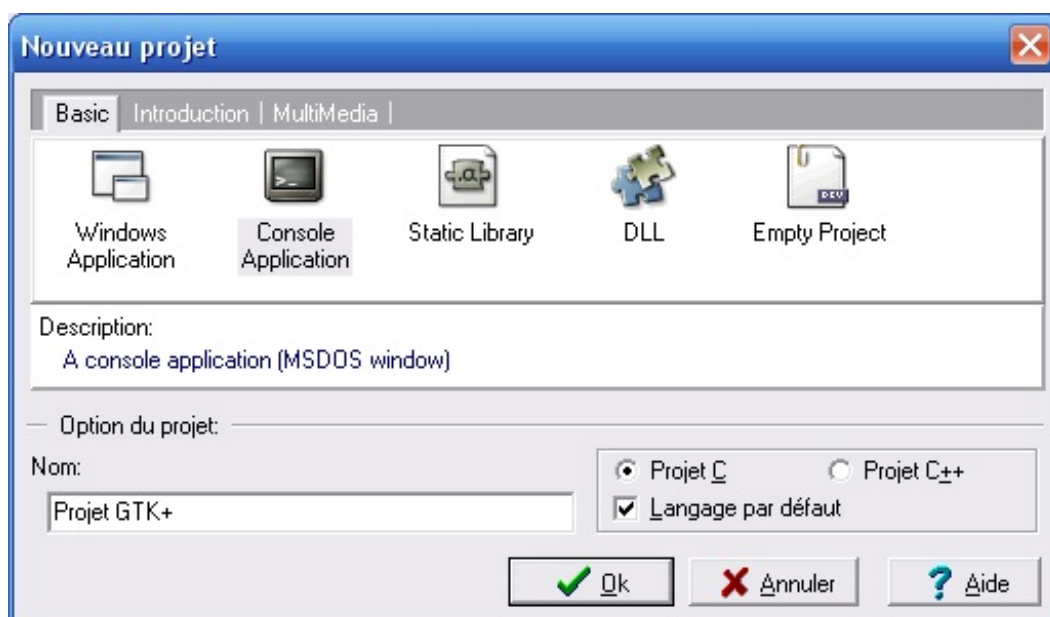
Configuration d'un Projet

Nous allons à présent créer et configurer un projet GTK+ 😊.

Comme pour la configuration de votre logiciel, il y a une explication pour chaque logiciel.



Créez un nouveau projet de type **Console Application** en C.



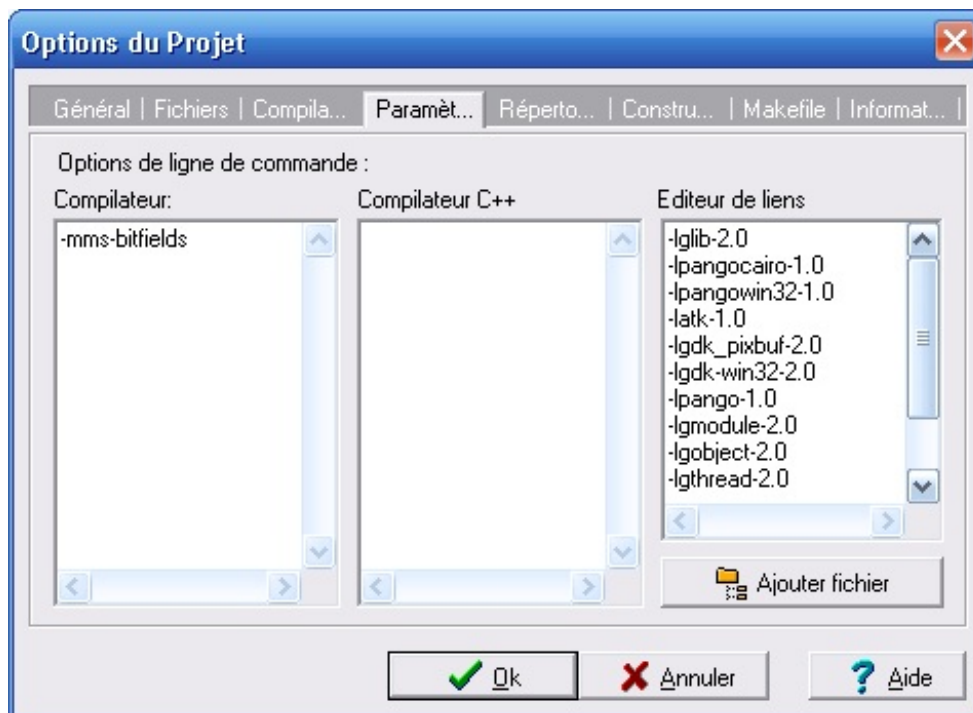
Ensuite, dans le menu, cliquez sur *Projet / Options du Projet*, puis allez dans l'onglet *Paramètres*.

Dans les Options du *Compilateur*, ajoutez :

-mms-bitfields

Et à droite, dans l'*Editeur de Liens*, ajoutez :

```
-lglib-2.0  
-lpangocairo-1.0  
-lpangowin32-1.0  
-latk-1.0  
-lgdk_pixbuf-2.0  
-lgdk-win32-2.0  
-lpango-1.0  
-lgmodule-2.0  
-lgobject-2.0  
-lgthread-2.0  
-lgtk-win32-2.0  
-lcairo
```



Puis cliquez sur *OK* pour appliquer les modifications.



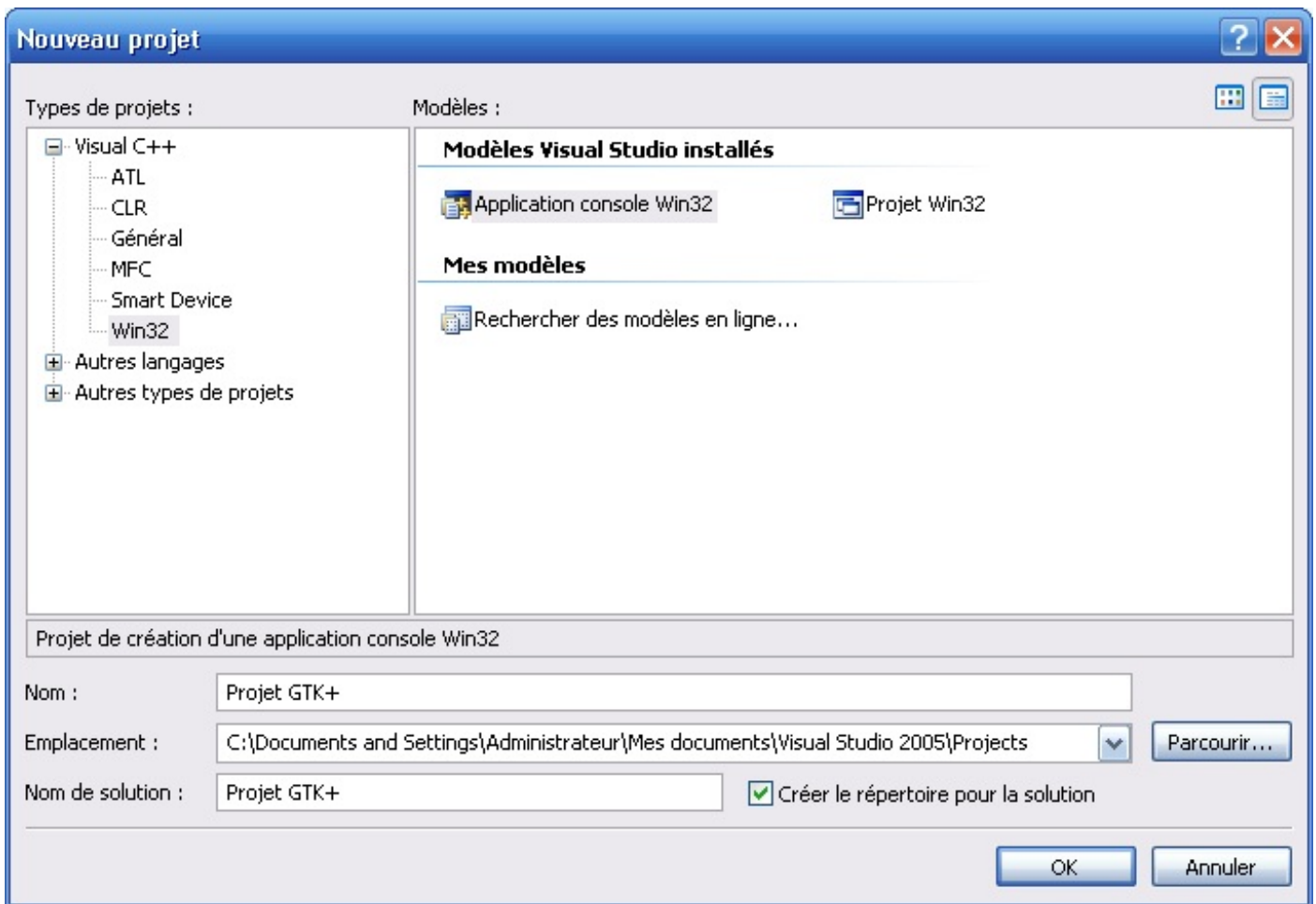
Ce projet étant un projet en console, la console s'affichera à l'exécution du programme.

Pour la cacher, allez dans le menu *Projet / Options du Projet* et dans l'onglet *Général*, à la partie *Type*, sélectionnez **Win32 GUI** puis appliquez la modification en cliquant sur *OK* 😊.



Pour Visual C++

Créez un nouveau projet de type **Application console Win32**.

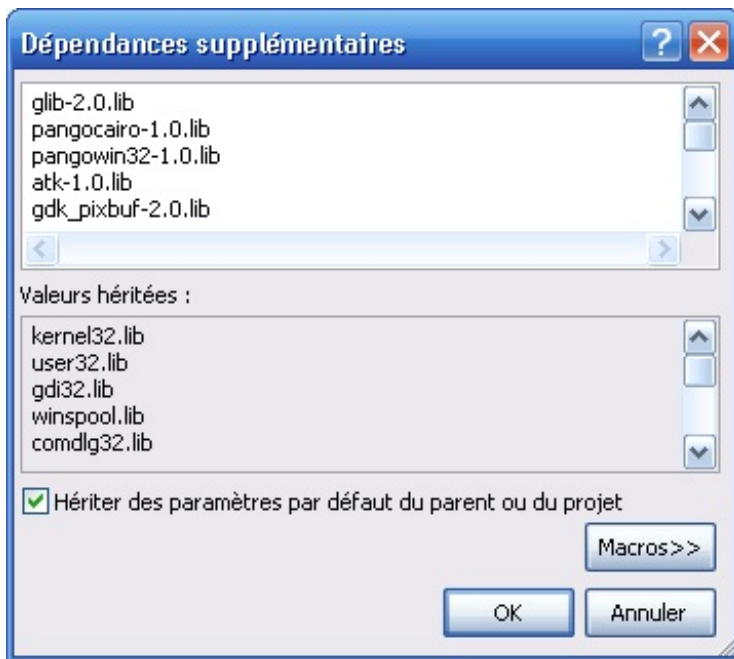


Ensuite, dans le menu, cliquez sur *Projet / Propriétés de <nom_du_projet>...*

A gauche, dans la sélection, allez dans *Propriétés de configuration / Editeur de liens / Entrée*.

Cliquez sur *Dépendances supplémentaires* puis sur les trois petits points qui apparaissent à droite. Dans la fenêtre qui s'ouvre, ajoutez :

glib-2.0.lib
pangocairo-1.0.lib
pangowin32-1.0.lib
atk-1.0.lib
gdk_pixbuf-2.0.lib
gdk-win32-2.0.lib
pango-1.0.lib
gmodule-2.0.lib
gobject-2.0.lib
gthread-2.0.lib
gtk-win32-2.0.lib
cairo.lib



Cliquez sur *OK*, puis sur *Appliquer* pour appliquer les modifications et sur *OK* pour fermer la fenêtre des propriétés du projet.



Si vous faites un Copier/Coller pour entrer ces paramètres, il est possible qu'une erreur de ce type survienne :

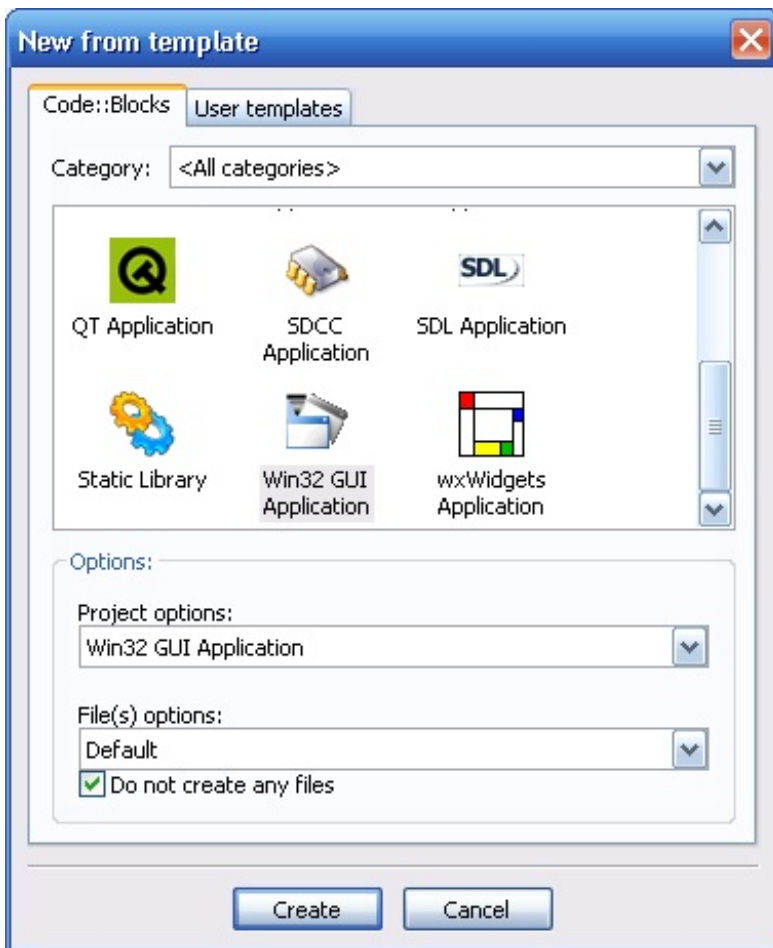
LINK : fatal error LNK1104: impossible d'ouvrir le fichier 'glib-2.0.lib pangocairo-1.0.lib'

Sachez que si elle apparaît, les paramètres du Projet n'ont pas été correctement entrés (copiez les lignes une par une pour être sûr de ne pas avoir de problèmes de retours à la ligne 😊).



Pour Code::Blocks

Créez un nouveau projet de type **Win32 GUI Application**.



Ensuite, dans le menu, cliquez sur *Project / Build options* et allez dans l'onglet *Compiler* puis dans *Other options* pour ajoutez : **-mms-bitfields**



Allez dans l'onglet *Linker* et dans la sélection de gauche (*Link librairies*), ajoutez des lignes dans lesquelles vous entrerez :

glib-2.0
 pangocairo-1.0
 pangowin32-1.0
 atk-1.0
 gdk_pixbuf-2.0
 gdk-win32-2.0
 pango-1.0
 gmodule-2.0
 gobject-2.0
 gthread-2.0
 gtk-win32-2.0
 cairo

(Il faut les faire une par une ! 😊)

Quand vous les avez toutes faites, vous devez voir ceci :



Puis cliquez sur *OK*.

Vous avez maintenant fini d'installer GTK+ ! 😊

Astuces

Cette Partie est consacrée à des astuces qui pourraient vous être très utiles ! 😊

Nouveau projet déjà configuré avec Dev-C++

Si vous avez Dev-C++ et que vous souhaitez ne pas à avoir à configurer tous vos projets GTK+, il existe une technique très simple permettant de gagner beaucoup de temps ! 😊

Créez un nouveau projet et configurez le. Il vous suffit maintenant de vous en servir comme original. Lorsque vous voudrez faire un nouveau projet GTK+, il vous suffira de copier le dossier contenant le projet déjà configuré et d'utiliser cette copie ! 😊

Nouveau projet déjà configuré avec Visual C++

Pour Visual C++, il faut modifier les ressources du logiciel pour que les *Dépendances supplémentaires* soient automatiquement mises dans les Options des Projets 😊.

Allez donc dans le dossier *VCProjectDefaults* qui se trouve dans les ressources du logiciel, par défaut ici :

C:\Program Files\Microsoft Visual Studio 8\VC\VCProjectDefaults

Ouvrez le fichier *corewin_express.vsprops* avec un éditeur de texte.

Remplacez la ligne 8 par :

```
AdditionalDependencies = "kernel32.lib glib-2.0.lib pangocairo-1.0.lib pangowin32-1.0.lib atk-1.0.lib gdk_pixbuf-2.0.lib  
gdk-win32-2.0.lib pango-1.0.lib gmodule-2.0.lib gobject-2.0.lib gthread-2.0.lib gtk-win32-2.0.lib cairo.lib"/>
```

Et n'oubliez pas d'enregistrer les modifications ! 😊

Maintenant, tout vos projets seront configurés pour fonctionner avec GTK+.

Nouveau projet déjà configuré avec Code::Blocks

Avec Code::Blocks, c'est encore plus simple : il est possible de sauvegarder la configuration d'un projet pour que lorsque vous

en créez un nouveau, il puisse être déjà configuré ! 😊

Voici comment faire:

Après avoir configuré le Projet, dans le menu, cliquez sur *Project/Save project as user-template*.

Dans la fenêtre qui s'ouvre, tapez:

Projet GTK+

et cliquez sur *OK*.

Maintenant, pour créer un nouveau projet déjà configuré, il vous suffira de cliquer sur l'icône *New Project* et dans l'onglet *User templates*, de sélectionner *Projet GTK+* puis de cliquer sur *Create*.

A Propos du Pack GTK+...

Vous pouvez tout à fait installer le Pack GTK+ ailleurs que dans **C:\Program Files\Pack GTK+** mais attention, vous devrez changer les chemins des répertoires des dossiers lorsque vous configurerez votre logiciel et votre projet 😊.

Vous pouvez à présent créer des applications GTK+ ! 😊



Pour vérifier que tout fonctionne, compilez et exécutez [ce code](#). Une fenêtre devrait s'ouvrir !

Si vous avez un problème à la compilation ou à l'exécution, vous avez peut-être mal installé GTK+, je vous conseille donc de vérifier que vous avez tout bien paramétré. Pour vérifier que ce tutoriel fonctionne, il a été testé plusieurs fois, donc ne me dites pas que ça ne marche pas ! 😊

L'installation de GTK+ sous Linux (Ubuntu & compagnie) est traitée dans le prochain chapitre.

By [Im@GinE](#)

[Picxime](#) & [Guimers8](#) remercient [Im@GinE](#) pour l'écriture de ce chapitre et pour la relecture du cours.


[Im@GinE](#) remercie [antoinexp](#) pour avoir testé l'installation.

[Im@GinE](#) remercie [Petrus6](#) pour l'astuce de `Code::Blocks` et pour avoir testé l'installation.

Installer GTK+ sous Linux


Après l'installation sous Windows, passons maintenant à l'installation sous Linux, car à la base GTK+ a été conçue pour Linux !




La compilation est expliquée en mode console avec [GCC](#) et ensuite dans la seconde partie, avec l'utilisation de Code::Blocks (qui a une interface graphique ).

Alors si vous êtes prêts, amis Linuxiens allons-y !



Si certains ont déjà installé GTK+ sous Windows il vont se rendre compte que l'installation sous Linux est beaucoup plus facile ! 

Télécharger et installer les fichiers

Tout d'abord il faut télécharger les bibliothèques qui vont permettre de compiler les programmes. Il vous faut simplement une connection internet, il ne devrait pas y avoir de problèmes puisque vous êtes sur internet ! 

Vous pouvez faire cela de deux façons : soit par la console soit par votre gestionnaire de paquets.

Mode Console

Je vais vous expliquer comment installer GTK+ entièrement par console pour les distributions Debian-based. Si vous avez une RedHat-like, par exemple Fedora, la commande ressemblera à ceci :

Code : Console

```
su -  
yum votre_commande
```

Nous allons commencer par mettre vos paquets à jour, ouvrez donc la console et tapez :


Code : Console

```
sudo apt-get update
```

Maintenant, installez le paquet de développement (qui contient les bibliothèques). C'est toujours aussi simple, il suffit de faire :

Code : Console

```
sudo apt-get install libgtk2.0-dev
```

Vous devriez normalement avoir tout ce qu'il faut pour développer avec GTK, mais il faut s'assurer que vous disposez bien du [Runtime](#), et comme prudence est mère de sureté nous allons vérifier ceci de suite .

Faites juste :

Code : Console

```
sudo apt-get install libgtk2.0-0
```

Si le système vous répond que vous l'avez déjà installé, vous n'avez plus qu'à programmer ! 😊

Mode Graphique

Avant de télécharger et d'installer les fichiers ressources, il faut que vos dépôts soient à jour. Lancez donc le **Gestionnaire de paquets** à partir du menu **Système** :



..... Ubuntu Xubuntu Kubuntu.

Dans la fenêtre qui s'ouvre, cliquez sur **Recharger** dans la barre d'outils, puis sur **Tout mettre à jour**.



Comme vous pouvez le voir sur les captures d'écran, selon la distribution de Linux, la façon de lancer le Gestionnaire des paquets est différente. Donc les explications ci-dessous peuvent aussi changer selon la distribution que vous avez, l'important est de faire ce qu'il est dit même si par exemple un bouton n'a pas le même nom que je le dis mais qu'il a la même fonction 😊.

Nous allons maintenant installer le paquet correspondant aux fichiers ressources. Dans la liste, recherchez un paquet nommé [libgtk2.0-dev](#).

Attention, prenez celui avec **-dev** à la fin, pas un autre !

Faites clic droit dessus, cliquez sur **Sélectionner pour installation** et si une fenêtre s'ouvre choisissez **Ajouter à la sélection**.

Enfin, cliquez sur **Appliquer** dans la barre d'outils et dans la fenêtre qui a pu s'ouvrir sélectionnez **A installer** et cliquez sur **Appliquer**. Vous devez maintenant patienter pendant le téléchargement et l'installation...

La bibliothèque GTK+ a besoin d'un ensemble de fichiers appelé [Runtime](#) pour fonctionner (en plus des fichiers ressources). Sous Linux il est représenté par le paquet [libgtk2.0-0](#), s'il est présent dans la liste et qu'il n'est pas installé, installez-le de la même manière que vous avez installé le paquet précédent.

Compilation en ligne de commande

Maintenant que vous avez installé les fichiers nécessaires, nous allons compiler ! 😊

Pour compiler en console, il faut passer des paramètres au compilateur. Nous allons voir cela tout de suite plus en détail avec le compilateur GNU/GCC.



Évitez de mettre des caractères spéciaux (à, é, è, ç, ', ", ^, etc...) ou des espaces dans vos noms de fichiers, dossiers ou exécutables, cela pourrait engendrer des erreurs à la compilation !

Commencez par créer un répertoire où vous stockerez tous vos programmes. Ensuite, téléchargez [ce fichier](#) (qui contient un code GTK+ de base, il ouvre une fenêtre avec un label) puis placez ce fichier dans le dossier que vous venez de créer.

Ouvrez la console et placez-vous dans votre dossier (utilisez les commandes **cd**, **cd..** et **ls** 😊) :

Code : Console

```
cd /home/le_chemin_de_votre_dossier_personnel/
```

Ensuite, compilez le code avec la commande suivante :

Code : Console

```
gcc $(pkg-config --libs --cflags gtk+-2.0) main.c -o nom_de_lexecutable
```

Le `main.c` correspond au nom de notre fichier (code source du programme) et `nom_de_lexecutable` est le nom que vous souhaitez donner à votre programme. Si une erreur survient, vérifiez que vous avez bien entré la commande et si le problème persiste, refaites la manipulation pour installer les fichiers, vous vous êtes peut-être trompé ! 😊

Pour finir, lancez le programme par l'explorateur de fichiers ou directement par la console :

Code : Console

```
./nom_de_lexecutable
```

En remplaçant `nom_de_lexecutable` par le nom que vous avez donné à votre programme.

Une belle fenêtre doit apparaître ! 😊

Compilation sous Code::Blocks

Après l'explication de la compilation en console, on arrive maintenant à la partie interface graphique ! 😊

Dans cette partie, je vais vous montrer comment utiliser et configurer le logiciel Code::Blocks qui est un IDE portable, Open Source et très pratique pour GTK+ 😊

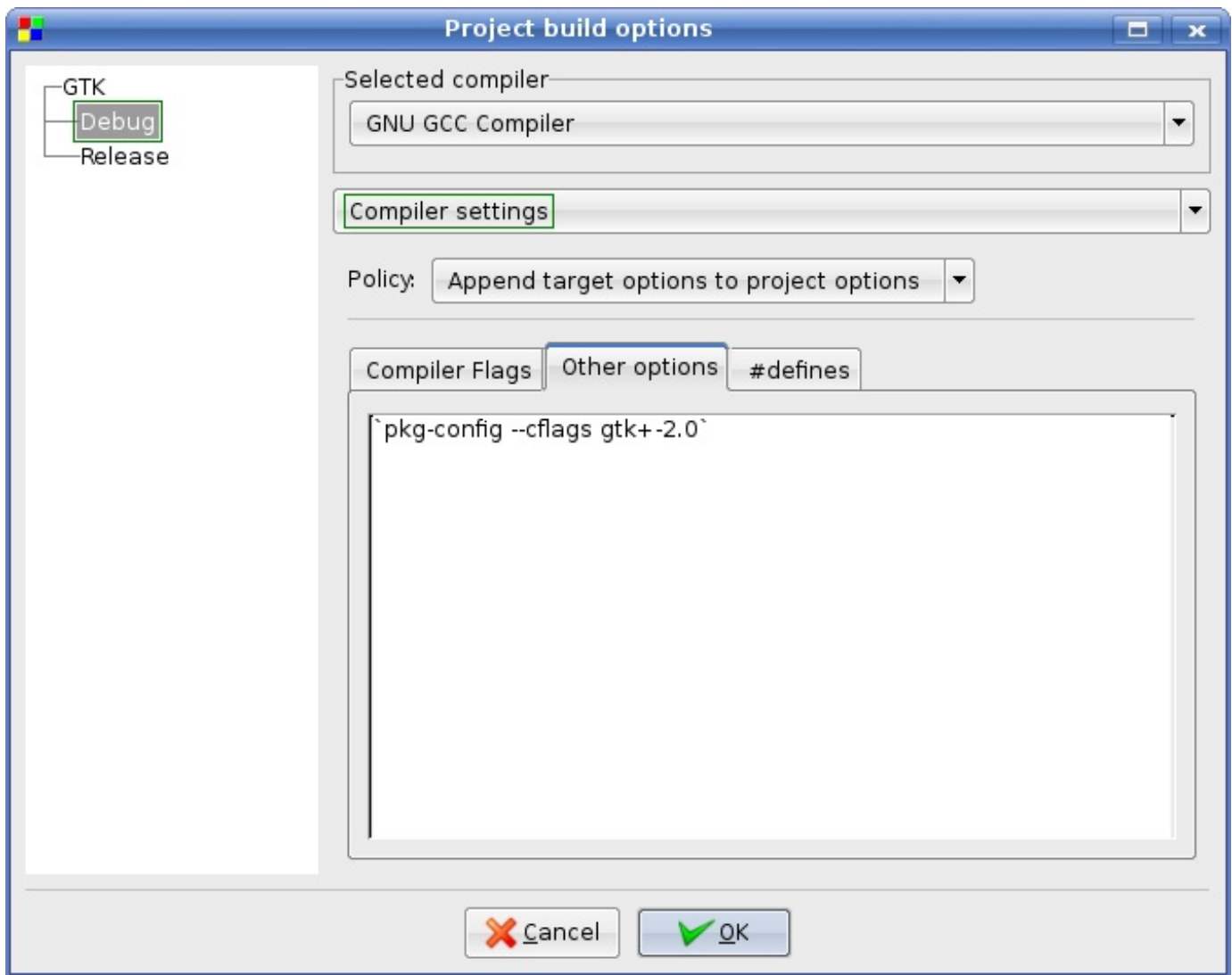
Configuration du logiciel

Lancez Code::Blocks et créez un nouveau Projet en faisant *File -> New Project -> Empty project*.

Ensuite, il est nécessaire d'indiquer au compilateur où il doit trouver les fichiers que vous avez installés dans la première partie. Pour cela, allez dans les *Options du Compilateur* (*Project -> Build Options*), partie *Debug* et dans *Other option*, entrez la ligne suivante :

Citation : Commande - Options du compilateur

```
`pkg-config --cflags gtk+-2.0`
```

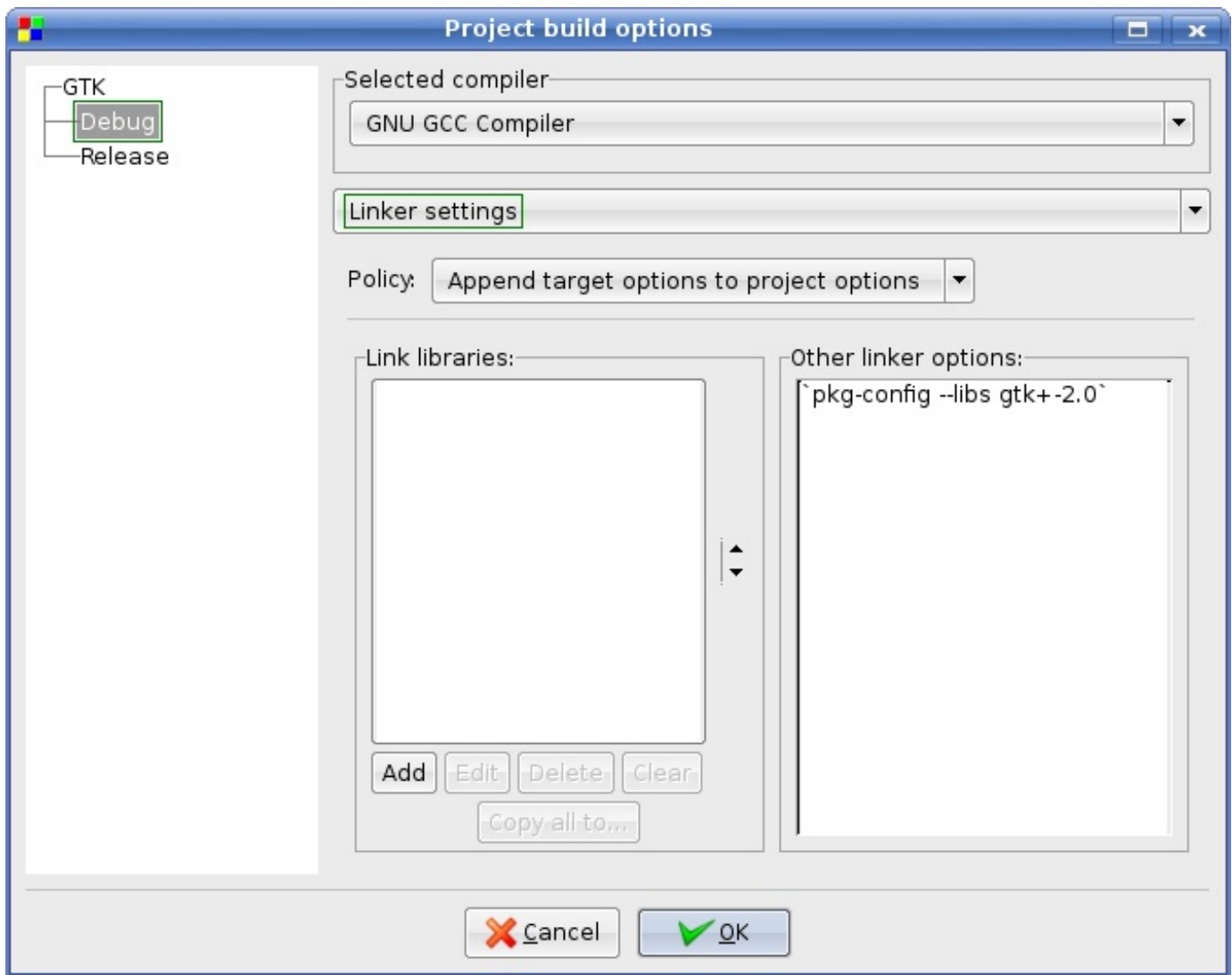


Selon la version que vous avez, l'interface peut varier, mais l'essentiel est de bien repérer la zone où doit se placer la commande 😊

Allez maintenant dans les *Options du Linker* et dans *Autre options (Other linker options)* ajoutez la ligne suivante :

Citation : Commande - Option du linker

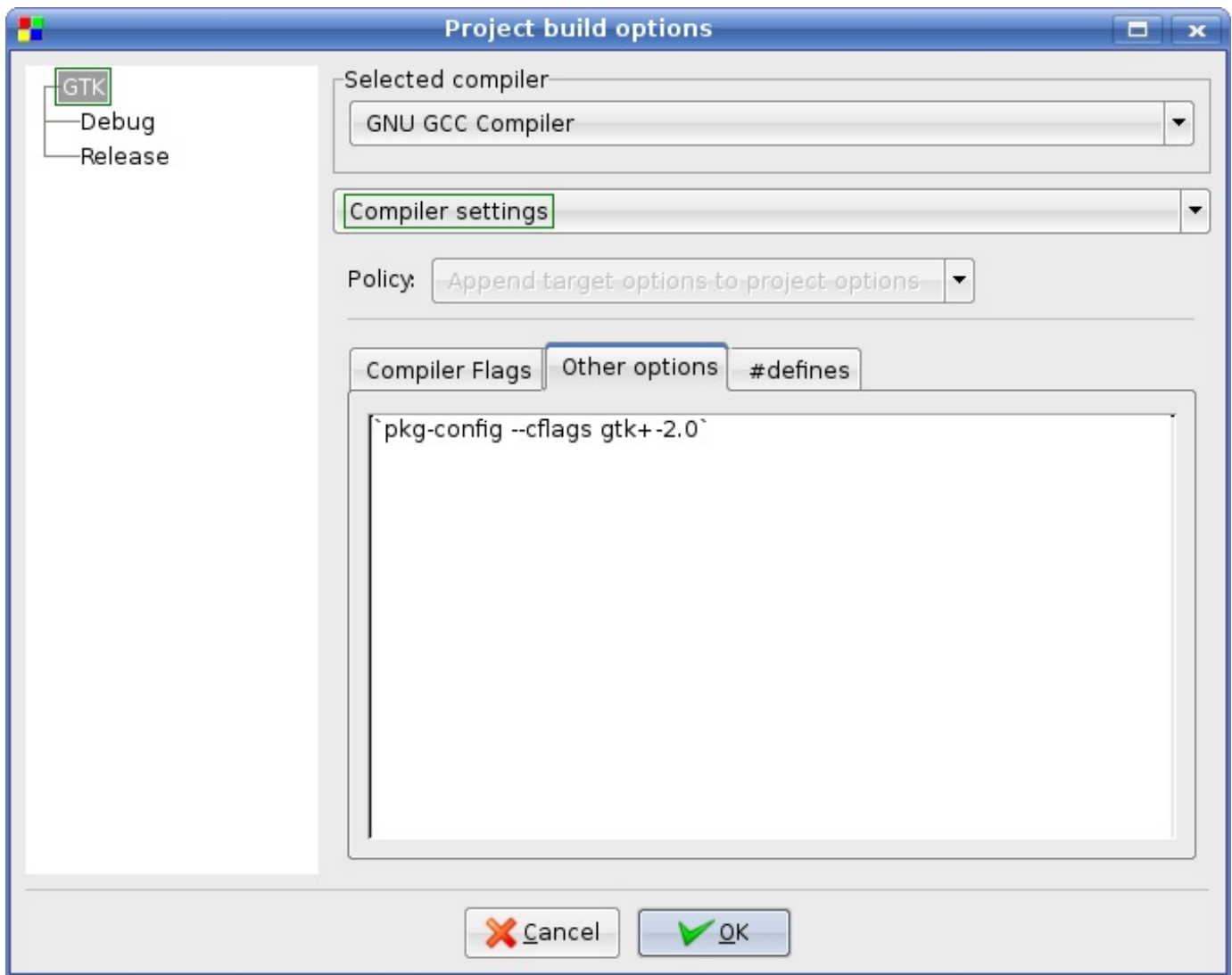
```
`pkg-config --libs gtk+-2.0`
```



Faites de même pour la partie qui doit porter le nom de votre projet *GTK* sur les screenshots.

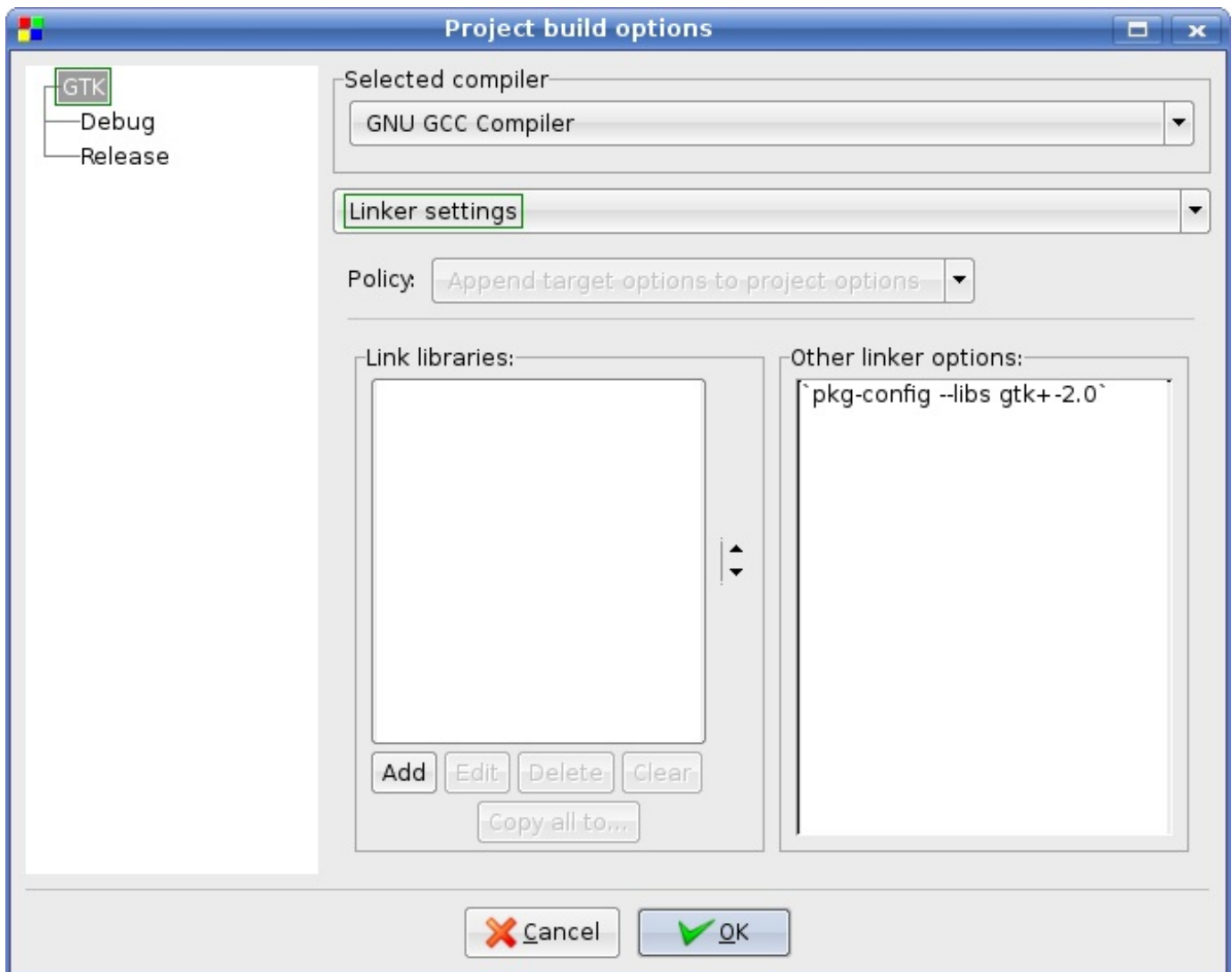
Citation : Commande - Options du compilateur

```
`pkg-config --cflags gtk+-2.0`
```



Citation : Commande - Option du linker

``pkg-config --libs gtk+-2.0``



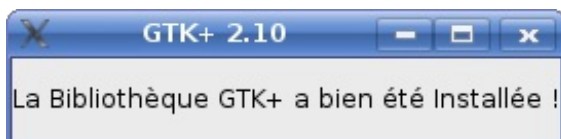
Encore une fois, l'interface peut varier.

C'est fini, il ne reste plus qu'à tester ! 😊

Test de la configuration

Nous allons vérifier que tout marche en compilant et exécutant un programme de base.

Téléchargez [le même code que pour la console](#), compilez-le et exécutez-le (F9), et une fenêtre devrait s'ouvrir :



Si c'est le cas, enregistrez votre projet, et régalez-vous avec GTK+ ! 😊

(Sinon vérifiez que vous ne vous êtes pas trompé durant la manipulation...)

Pour enregistrer toute la configuration, faites *Edit -> Save as user template*. De cette façon, il vous suffira de sélectionner votre template à chaque nouveau projet et votre logiciel sera configuré ! 😊

Ce chapitre touche à sa fin, j'espère qu'il vous a été utile et qu'il vous aura permis de faire ce que vous souhaitiez ! 😊

L'installation sous Mac est expliqué dans le chapitre suivant.

By [Petrus6](#)

JRG Soft remercie Petrus6 pour nous avoir offert ce très bon Tuto ! 😊

Et Petrus6 le remercie d'avoir fait une place pour ce Tuto.

*Petrus6 remercie [Im@GinE](#) pour ses conseils et pour l'avoir aidé,
[Guimers8](#) et [Picxime](#) pour la grande relecture, ainsi que
les trois Bêta-testeurs [Alexfun13](#), [Yno](#) et [Seb2003](#).*

Installer GTK+ sous MacOS X

Dans la lignée des chapitres d'installation, passons maintenant au cas *MacOS X* ! 😊 L'installation sur ce système est somme toute sensiblement similaire à celle qui existe pour les autres Unixöides, à quelques détails près bien entendu 😊.

Fink, X11 ?



L'utilisation de GTK+ sous MacOS X est légèrement différente des autres systèmes, du fait que son utilisation n'est pas tout à fait transparente pour l'utilisateur. En effet, si vous utilisez [The Gimp](#) (Photoshop libre), ou [Inkscape](#) (dessin vectoriel), vous voyez l'icône de X11 apparaître dans le dock avant que l'application ne se lance.

Cela est dû au fait qu'il n'existe pas, à l'heure actuelle, de portage natif de GTK+ utilisant les APIs de la pomme : Cocoa et Carbon. De ce fait, GTK+ est obligée d'utiliser le serveur X11 (XFree86) pour l'affichage, exactement comme elle le fait sous Linux.

Puisqu'il n'y a pas de version Mac de GTK+, comment va t-on faire pour compiler nos programmes ?

La solution est simple, et elle s'appelle **Fink**. Il s'agit d'un projet destiné à porter simplement des applications et paquets écrits à la base pour GNU/Linux vers MacOS X. C'est un gestionnaire de paquets, comme **dpkg** sous *Debian*, qui va nous permettre d'installer GTK+ et ses dépendances, notamment les paquets de développement.



Installation

Nous allons installer ce qui est nécessaire à GTK+ : XCode, X11, et Fink.

XCode

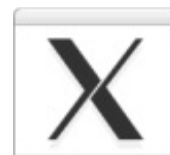


Je vous conseille fortement de programmer avec XCode, l'IDE à la pomme étant très puissant, mais pourtant très simple à utiliser. Il est livré avec Tiger sur le CD n°1, pour les autres versions, rendez-vous sur le site d'Apple, section [Téléchargements](#).

Dans tous les cas, vous devez impérativement posséder la version la plus récente tolérée par votre système (sinon cela ne marche pas).

X11

Si vous n'avez pas encore X11 sur votre machine, il est nécessaire de l'installer. Pour savoir si vous l'avez, allez dans *Applications/Utilitaires* et cherchez une icône blanche avec un *X* à l'intérieur.



MacOS X Tiger

Munissez-vous du CD d'installation n°1 de Tiger et insérez-le dans votre lecteur. Ouvrez l'installateur **Optional Installs**. Allez jusqu'à la sélection du type d'installation, et cochez la case **X11** (voir). Finalisez l'installation pour installer X11.

MacOS X Panther

Pour cette version de MacOS (et les versions inférieures), rendez-vous sur le site d'Apple et [téléchargez X11](#). Installez-le ensuite.



Lorsque vous distribuerez vos programmes, n'oubliez pas de préciser qu'ils nécessitent X11. Éventuellement, donnez un lien vers le site d'Apple ou indiquez la marche à suivre pour l'installer 😊.

Fink

Téléchargez la version de Fink correspondant à votre système :

- [10.4 Tiger Intel - PowerPC](#)
- [10.4 Tiger PowerPC](#)
- [10.3 Panther](#)

Ouvrez l'archive téléchargée, et lancez l'installateur. À la fin, il vous demande s'il peut exécuter un script, acceptez en choisissant **Oui**. Après l'installation, ouvrez un Terminal (*Applications/Utilitaires*) et installez les paquets suivants :

- gtk+2 (et gtk+2-shlibs)
- gtk+2-dev
- glib2 (et glib2-shlibs)
- glib2-dev
- pango (et pango-shlibs)
- pango-dev
- atkl (et atkl-shlibs)

Pour installer un paquet, tapez la commande :

Code : Console

```
fink install nom_du_paquet
```

Les paquets en parenthèses devraient s'installer automatiquement. Faites-le manuellement si ce n'est pas le cas.

Pour finir l'installation, toujours dans la console, mettez à jour tous les paquets installés en faisant :

Code : Console

```
fink update-all
```

Tout est maintenant installé ! 😊

Compilation

Ouvrez XCode, et créez un nouveau projet **Standard Tool** (catégorie **Command Line Utility**).

Configuration du projet

Les bibliothèques

Dans la colonne de gauche, on peut voir les dossiers *Source*, *Documentation* et *Products*. Cliquez-droit sur le nom du projet, et créez un nouveau dossier (**Add > New Group**) appelé **GTK Libs** (par exemple). Faites clique-droit sur ce dossier, puis choisissez **Add > Existing Files**. Là, sélectionnez les fichiers suivants, dans le dossier /sw/lib/ (attention aux noms qui se ressemblent) :

- libatk-1.0.0.dylib
- libgdk-x11-2.0.0.dylib
- libgdk_pixbuf-2.0.0.dylib
- libglib-2.0.0.dylib
- libgmodule-2.0.0.dylib
- libgobject-2.0.0.dylib
- libgthread-2.0.0.dylib

- libgtk-x11-2.0.0.dylib
- libpangox-1.0.0.dylib

Ces bibliothèques doivent donc se retrouver dans le dossier *GTK Libs* (voir).

Les fichiers d'en tête

Faites un clique-droit encore une fois sur le nom du projet, et allez sur **Get Info**, puis dans l'onglet **Build**.

Repérez la ligne **Header Search Paths**, et collez ceci comme valeur (à droite) :

Code : Console

```
/usr/include /sw/include/glib-2.0 /sw/lib/glib-2.0/include /sw/include/gtk-2.0 /sw/lib/gtk-2.0/include /sw/include/atk-1.0 /sw/include/pango-1.0 /sw/include/cairo
```

Faites de même pour la ligne **Library Search Paths** :

Code : Console

```
/sw/lib/ /sw/lib/glib-2.0 /sw/lib/gtk-2.0
```

Voilà pour ce qui est de la configuration, maintenant on compile ! 🤖

Compilation et exécution

Ouvrez le fichier main.c (ou autre), et entrez un code GTK+, à tout hasard :

Code : C

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
    /* Initialisation de GTK+ */
    gtk_init(&argc, &argv);
    return EXIT_SUCCESS;
}
```

Appuyez sur le bouton **Build** pour compiler votre projet.

Pour l'exécuter, cela va être un peu différent par rapport à d'habitude. En effet, le fait que GTK+ nécessite X11 pose quelques problèmes avec XCode, et on ne peut donc pas utiliser le bouton **Build & Run**.

Pour lancer le programme, on tapera dans un Terminal :

Code : Console

```
open-x11 ~/Desktop/Mon_Projet/build/Debug/mon_executable
```

Ce qui aura pour effet de lancer X11 si ce n'est déjà fait, puis votre programme 😊.

Génial, non ? 🧙

Vous êtes maintenant prêt à programmer ! 😊

Dans le chapitre suivant, vous allez apprendre les bases de GTK+ ainsi que la création d'une fenêtre.

By [Guimers8](#)

Notions de base et fenêtres

Vous voici donc dans le premier chapitre *intéressant* ! 😊

Dans un premier temps nous allons faire un peu de théorie pour comprendre comment GTK fonctionne, puis nous allons découvrir le code de base d'une application GTK.

Ensuite, nous allons voir comment créer une fenêtre, notamment la fenêtre principale de votre programme puis comment la personnaliser et la caractériser. En ce qui concerne les boîtes de dialogue (erreurs, infos, questions, etc), nous les verrons dans la partie II quand vous en saurez un peu plus. 😊

On a du pain sur la planche ! Bon courage et bonne lecture !

Widgets et héritage

Les Widgets

Avant de passer à la pratique, il va falloir que je vous explique certaines choses sur le fonctionnement des objets GTK (boutons, menus, etc...).

Lorsque vous développerez des applications avec GTK, les *objets* que vous mettrez dans vos fenêtres (je ne sais pas trop comment les appeler autrement que par leur vrai nom) sont appelés WIDGETS. Il n'existe pas de sens exact pour ce mot (sens général, bien sûr), mais en cherchant un peu, vous trouverez différentes définitions, voici quelques explications :

- De l'anglais : *machin, chose, gadget*.
- Gadget de Dashboard sous mac OSX Tiger.
- Plus intéressant : **WInDow gadGET** ou **élément d'une interface graphique**.

Je pense que vous commencez à comprendre de quoi il s'agit !!! 😊 En gros, les widgets sont les boutons, les zones de texte, les menus, enfin.. à peu près tout ce qui constitue une interface !

Quelques bases

- Il existe des types différents (ce sont des structures) pour chaque type de widget. Cependant, on ne les utilise que dans des cas particuliers et on utilisera la plupart du temps le type "widget" pour stocker les objets.
- Les noms des fonctions ont une syntaxe particulière, ils sont de la forme **gtk_widget_action(...)**. C'est cela dans la plupart des cas avec GTK.

On remplace :

- *widget* par le type de widget sur lequel on travaille (ex : window pour une fenêtre).
- *action* par ce que la fonction est sensée effectuer (ex : set_title pour définir un titre)

C'est très facile à comprendre, vous ne devriez pas avoir de problème avec ça ! 😊

L'héritage

Après les Widgets, il y a une autre notion à connaître avant de commencer, l'héritage ! Si vous faites du C++, vous devriez savoir ce que c'est car c'est le but du C++ : l'héritage et l'orienté objet ! Mais nous travaillerons en C et GTK introduit une notion d'héritage.

Il faut que vous sachiez qu'il existe une hiérarchie entre les différents Widgets : il y a des groupes qui contiennent plusieurs widgets, la possibilité qu'un widget en contienne un autre, etc. L'héritage de GTK y est directement lié !

Un exemple concret

Je vais vous expliquer l'héritage avec un exemple : Une fenêtre.

Je vous ai expliqué que pour créer un objet, on déclare un widget puis on y stocke un objet quelconque.
Une fenêtre est donc avant tout un Widget.

Citation : Hiérarchie des objets

Widget -> fenêtre

Maintenant, imaginons que je veuille créer une boîte de dialogue. Une boîte de dialogue c'est quoi ? En y réfléchissant un peu, on s'aperçoit que c'est aussi une fenêtre ! On y a apporté des modifications mais c'est une fenêtre. On va donc descendre d'un cran pour dire qu'elle est particulière :

Citation : Hiérarchie des objets

Widget -> Fenêtre -> Boîte_de_dialogue

Petit point de vocabulaire : On dit que *BOITE_DE_DIALOGUE* **dérive de** *FENETRE* (retenez bien le terme de dériver). Dans la réalité, l'architecture est bien plus compliquée, mais d'un point de vue théorique, c'est exactement ça !

A quoi ça sert ?

L'intérêt de tout ça ? Il est simple.

Un premier exemple : Pour créer une boîte de dialogue, les développeurs de GTK se sont dit : "On a créé une fenêtre, y'a plus qu'à la modifier !". La fonction qui crée une boîte de dialogues appelle la fonction qui crée une fenêtre, puis modifie certains paramètres pour en faire une boîte de dialogues.

Un peu de pseudo-code C :

Code : C

```
Widget* nouvelle_boite_dialogue(void)
{
    Widget * fenetre = nouvelle_fenetre();

    /* On change la taille, on ajoute du texte et des boutons, ..
    pour faire une boîte de dialogue */

    return fenetre;
}
```

Un deuxième exemple : Les développeurs de GTK ont programmé une fonction pour définir le titre d'une fenêtre. Mais pour une boîte de dialogue ?

Se sont-ils *ennuyés* à refaire la même fonction (car une boîte de dialogue a la même structure qu'une fenêtre) pour une boîte de dialogue ? Et donc avoir deux fonctions identiques pour deux objets similaires ? ... NON !

Tout simplement, vous utiliserez sur *BOITE_DE_DIALOGUE* une fonction faite pour *FENETRE*, ce qui ne pose aucun problème puisque *BOITE_DE_DIALOGUE* **dérive de** *FENETRE*.

Concluez que vous pouvez utiliser sur un widget les fonctions des widgets parents. 😊 (dans la plupart des cas)

Voilà à quoi sert l'héritage, à minimiser le nombre de fonctions pour des choses identiques. Si, ça reste assez flou pour vous, méditez bien là-dessus jusqu'à ce que vous ayez bien compris, c'est très important ! 😊

Code de base et création d'une fenêtre

Code de base GTK

Dans le chapitre précédent, vous avez installé la bibliothèque GTK, les runtimes nécessaires pour que vos programmes puissent être exécutés, et même d'autres bibliothèques permettant à GTK de faire des choses beaucoup plus avancées ! (Car la bibliothèque GTK toute seule ne sert pas à grand chose 😊)

Après (peut-être) quelques difficultés, vous êtes arrivé à la fin de ce chapitre où vous avez trouvé le code de base d'une application GTK, afin de tester si tout marchait bien ; je vous le remet ici :

Code : C

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
    /* Initialisation de GTK+ */
    gtk_init(&argc, &argv);
    return EXIT_SUCCESS;
}
```

Citation : M@teo

Un grand nombre de bibliothèques écrites en C nécessitent d'être initialisées et fermées par des appels à des fonctions. La SDL n'échappe pas à la règle.

GTK+ non plus ! 😞 En fait, GTK+ a juste besoin d'être initialisé, pas besoin de la quitter.

Vous remarquez donc les nouveautés suivantes :

- **int main(int argc, char **argv)** : En fait, rien de nouveau mais n'oubliez pas que *le main doit avoir ce prototype* !
- **#include <gtk/gtk.h>** : L'inclure de la bibliothèque GTK+, charge la bibliothèque et permet l'utilisation des fonctions GTK & co.
- **gtk_init(&argc, &argv)** : Première fonction, elle initialise GTK et charge certaines choses en mémoire. On lui passe l'adresse des arguments du main.
- **return EXIT_SUCCESS** : EXIT_SUCCESS est, disons, *mieux* que 0. C'est donc cela que nous utiliserons. 😊

Si vous compilez, vous remarquerez que rien n'apparaît à l'écran, normal, tout comme la SDL, initialiser la bibliothèque ne suffit pas, il faut créer une fenêtre. 😞 (ou autre chose, mais on va commencer par ça !)

Création d'une fenêtre

Faites bien attention, la méthode que je vais utiliser pour vous expliquer les fenêtres sera la même que celle que j'utiliserai tout au long du cours.

Cela se passe en trois étapes :

- Explication du widget, hiérarchie et fonctionnement.
- Fonctions relatives à ce/ces Widget(s).
- Code d'exemple complet.

Le nom du widget fenêtre est **GtkWindow**. Voici sa hiérarchie :

Citation : Hiérarchie de GtkWindow

GObject -> GtkWidget -> GtkWidget -> GtkContainer -> GtkBin -> GtkWindow

Vous allez donc déclarer **un pointeur sur GtkWidget**, vous pouvez le déclarer au début avant l'initialisation. Donnez lui un nom simple et clair, ce sera votre fenêtre principale ! :

Code : C

```
GtkWidget * MainWindow = NULL;
```

Ensuite, vous allez assigner une nouvelle fenêtre à ce pointeur avec la fonction `gtk_window_new` qui prend en paramètre le type de fenêtre à créer.

Code : C

```
GtkWidget* gtk_window_new(GtkWindowType type);
```

Les deux types possibles sont :

- `GTK_WINDOW_TOPLEVEL` : Une fenêtre normale.
- `GTK_WINDOW_POPUP` : Une fenêtre sans bordure (ne pas utiliser, sauf cas particuliers).

Vous allez donc faire :

Code : C

```
MainWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

On dit que `gtk_window_new` est le constructeur de `GtkWindow`.



Vous pouvez voir que ça fait à peine 2 minutes qu'on a commencé et qu'on a déjà pas mal d'anglais avec tous ces mots, *new*, *window*, *oplevel*, etc.. ! Parlons des anglophobes, il vous faudra inévitablement apprendre les bases pour pouvoir étudier les docs, le site du Zéro ne peut pas être toujours derrière vous pour tout vous traduire. De plus, les noms, mots et expressions utilisés dans les noms de fonctions et arguments vous paraîtront bien plus clairs comme ça et ce sera plus facile à retenir ! 😊

Veuillez m'excuser pour ce petit hors-sujet.

Afficher la fenêtre

Pour afficher un widget, on utilise la fonction `gtk_widget_show`, quel que soit le widget !

Code : C

```
void gtk_widget_show(GtkWidget *widget);
```

Plus tard vous pourrez aussi utiliser `gtk_widget_show_all`, cette fonction permet de montrer tous les widgets qui ont été insérés dans un GtkContainer (`GtkWindow` dérive de `GtkContainer`) :

Code : C

```
void gtk_widget_show_all(GtkWidget *widget);
```

Evènements

Un cours sera dédié à la programmation événementielle mais il va bien falloir que je vous donne un bonus en attendant. Vous

allez donc me faire confiance et rajouter cette ligne après la création de la fenêtre :

Code : C

```
g_signal_connect(G_OBJECT(MainWindow), "delete-event",
G_CALLBACK(gtk_main_quit), NULL);
```

Et juste avant le return, vous rajoutez cet appel de fonction :

Code : C

```
gtk_main();
```

Bon, vous l'avez mérité, voilà le code complet. Essayer d'apprendre le nom des fonctions et de ne pas faire trop de copier/coller !



Code : C

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
    /* Variables */
    GtkWidget * MainWindow = NULL;

    /* Initialisation de GTK+ */
    gtk_init(&argc, &argv);

    /* Création de la fenêtre */
    MainWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    g_signal_connect(G_OBJECT(MainWindow), "delete-event",
G_CALLBACK(gtk_main_quit), NULL);

    /* Affichage et boucle événementielle */
    gtk_widget_show(MainWindow);
    gtk_main();

    /* On quitte.. */
    return EXIT_SUCCESS;
}
```

Compilez et exécutez !!! Voilà votre première fenêtre qui s'ouvre sous vos yeux, quelle émotion, ..., non ? 😊

Bon; c'est pas super beau, c'est vide, mais bon c'est une fenêtre ! C'est tout ce qu'on a demandé !

Nous allons maintenant l'améliorer, lui donner un titre, une taille, et la mettre où on veut ! 😊

Personnaliser notre fenêtre

La partie que vous attendiez, on va pouvoir s'amuser ! 😄

Je vais vous présenter pleins de fonctions intéressantes pour vos fenêtres. Certaines ne vous serviront pas pour le moment mais si vous en avez besoin, vous savez qu'elles sont là !



La quasi-totalité de ces fonctions vous demanderont un *GtkWindow* à la place d'un *GtkWidget*. La solution consiste à utiliser la macro **GTK_NOM_DU_WIDGET**, autrement dit, **GTK_WINDOW()**. N'oubliez pas, car sinon, vous aurez des erreurs dans la console (c'est pas beau, et ça peut créer des super bugs incompréhensibles).

Le titre

Fonction pour définir le titre : on donne la fenêtre et une chaîne de caractères :

Code : C

```
void gtk_window_set_title(GtkWindow *window, const gchar *title);
```

Il faut donc faire ainsi :

Code : C

```
gtk_window_set_title(GTK_WINDOW(MainWindow), "Fenetre de zer0");
```

Fonction pour récupérer le titre : La fonction retourne un *gchar**. C'est le char de la GLib, je vous conseille ce type pour utiliser Gtk, sinon vous aurez toujours pleins de Warnings ! 😊

Code : C

```
const gchar* gtk_window_get_title(GtkWindow *window);
```

Attention, vous devez bien stocker le retour de cette fonction dans un **const gchar**, car le pointeur retourné pointe directement là où GTK+ stocke le titre de la fenêtre : vous ne devez par conséquent pas libérer cette chaîne.

La taille (par défaut)

Définir la taille par défaut : On donne la fenêtre (avec la macro) et les dimensions voulues.

Code : C

```
void gtk_window_set_default_size(GtkWindow *window, gint width, gint height);
```

Recupérer la taille par défaut : On donne deux pointeurs *gint* (int de la Glib, comme le *gchar* !) et Gtk les modifie pour vous donner la taille. 😊

Code : C

```
void gtk_window_get_default_size (GtkWindow *window, gint *width, gint *height);
```

La taille par défaut sera utilisée si elle est assez grande pour contenir tous les widgets, sinon, elle sera agrandie automatiquement.

La taille (actuelle)

Définir la taille actuelle :

Code : C

```
void gtk_window_resize (GtkWindow *window, gint width, gint height);
```

Récupérer la taille actuelle :

Code : C

```
void gtk_window_get_size (GtkWindow *window, gint *width, gint *height);
```

Les paramètres sont les mêmes que pour la taille par défaut.

Positionnement

Fonction pour positionner la fenêtre : On passe un *GtkWindowPosition*. Il y a les possibilités suivantes :

- GTK_WIN_POS_NONE : Position aléatoire.
- GTK_WIN_POS_MOUSE : Position de la souris au moment de l'appel.
- GTK_WIN_POS_CENTER : Fenêtre centrée.
- GTK_WIN_POS_CENTER_ALWAYS : Toujours centrée, non déplaçable.
- GTK_WIN_POS_CENTER_ON_PARENT : Centrée par rapport à la fenêtre parente.

Code : C

```
void gtk_window_set_position(GtkWindow *window, GtkWindowPosition position);
```

Ou avec la position en *x* et *y* :

Code : C

```
void gtk_window_move(GtkWindow *window, gint x, gint y);
```

Fonction pour récupérer la position : Voir la fonction pour la taille.

Code : C

```
void gtk_window_get_position(GtkWindow *window, gint *root_x, gint *root_y);
```

L'icône

Définir l'icône : On donne la fenêtre, le nom du fichier, et un pointeur sur *GError* pour gérer les erreurs (ou NULL). La fonction renvoie TRUE si l'icône a pu être définie à la fenêtre.

Code : C

```
gboolean gtk_window_set_icon_from_file (GtkWindow *window, const
gchar *filename, GError **err);
```

Vous pouvez aussi définir l'icône à partir du `GdkPixbuf`, avec la fonction `gtk_window_set_icon`.

Récupérer l'icône : donnez juste votre fenêtre, la fonction renvoie un `GdkPixbuf`.

Code : C

```
GdkPixbuf * gtk_window_get_icon (GtkWindow *window);
```

Comme son nom l'indique, le *GdkPixbuf* est un buffer de pixel, on l'utilise pour manipuler les images avec GTK+. Nous apprendrons à nous en servir plus tard.

Iconifier

Iconifier :

Code : C

```
void gtk_window_iconify (GtkWindow *window);
```

Restaurer :

Code : C

```
void gtk_window_deiconify (GtkWindow *window);
```

Maximiser

Maximiser :

Code : C

```
void gtk_window_maximize (GtkWindow *window);
```

Restaurer :

Code : C

```
void gtk_window_unmaximize (GtkWindow *window);
```

Bouton "fermer"

Définir l'état du bouton fermer : Donnez TRUE pour activer le bouton, FALSE pour le désactiver.

Code : C

```
void gtk_window_set_deletable (GtkWindow *window, gboolean setting);
```

Connaître l'état : La fonction renvoie TRUE si le bouton est actif, FALSE sinon.

Code : C

```
gboolean gtk_window_get_deletable (GtkWindow *window);
```

Les bordures

Définir l'épaisseur des bordures : Donnez la fenêtre et l'épaisseur des bordures en pixels, dans l'ordre *gauche, haut, droite, bas*.

Code : C

```
void gtk_window_set_frame_dimensions (GtkWindow *window, gint left,
gint top, gint right, gint bottom);
```

Connaître l'épaisseur : Mêmes paramètres, mais en pointeurs pour que vous puissiez y avoir accès.

Code : C

```
void gtk_window_get_frame_dimensions (GtkWindow *window, gint *left,
gint *top, gint *right, gint *bottom);
```

Exemple

Pour que vous compreniez bien l'utilisation de ces fonctions, voici un exemple de code personnalisant une Fenêtre, il utilise plusieurs fonctions vue ci-dessus : [Télécharger le code exemple](#).

Vous savez maintenant créer des fenêtres GTK+ et les personnaliser ! 😊

Dans le prochain chapitre, vous allez apprendre à utiliser les labels pour pouvoir afficher du texte dans vos fenêtres ! 😊

Enjoy !

By [Guimers8](#)

Le texte avec les labels

Dans le chapitre précédent, nous avons vu comment faire une fenêtre. Le problème est qu'une fenêtre vide sert à rien. C'est pourquoi, dans ce chapitre, nous allons découvrir comment afficher du texte dans nos fenêtres. Pour cela nous allons utiliser des labels. Pour ceux qui ne parlent pas anglais, label veut dire étiquette. Nous allons donc "accrocher" des étiquettes sur nos widgets :p.

lais signifiant étiquette.

Les GtkLabels

Créer un label

Tout d'abord, nous allons déclarer un nouvel objet **GtkLabel** de type GtkWidget. Vous pouvez l'appeler comme vous le voudrez. Je rappelle qu'il est indispensable de déclarer un widget avant de l'utiliser.

Code : C

```
GtkWidget* label = NULL;
```

Maintenant que notre widget est déclaré, nous allons l'initialiser avec la fonction suivante:

Code : C

```
GtkWidget* gtk_label_new (const gchar *str);
```

Le paramètre :

- **str** : Le texte de notre label entre guillemets.

Voilà, notre **GtkLabel** est créé. Nous allons maintenant le rajouter dans la fenêtre avec la fonction suivante :

Code : C

```
void gtk_container_add (GtkContainer *container, GtkWidget *widget);
```

Les paramètres :

- **container** : Le nom du widget qui contiendra notre label. Dans notre cas, c'est la fenêtre.
- **widget** : Le nom du widget à ajouter. Ici, nous rajouterons notre label



Attention, **container** est de type **GtkContainer**. Pour que notre fonction marche nous allons utiliser une macro comme pour les fenêtres: **GTK_CONTAINER()**.

N'oubliez pas d'afficher la fenêtre et le label, pour ce faire utilisez la fonction **gtk_widget_show_all** ! 😊



Les deux dernières fonctions sont très très importantes, elles vous serviront dans tous vos programmes. La première sert à ajouter un widget dans la fenêtre (ou dans un autre container) et la deuxième sert à afficher cette fenêtre. Dans la suite de ce tutorial, je ne vais pas répéter qu'il faut utiliser ces fonctions à chaque fois. C'est logique, sans ces fonctions, rien ne s'affiche.

Récupérer et modifier un label

Récupérer un label

Dans certains programmes, il est parfois nécessaire de récupérer le contenu d'un label. Là aussi, une fonction très simple :

Code : C

```
const gchar* gtk_label_get_label (GtkLabel *label);
```

Le paramètre :

- **label** : Le nom de notre label que l'on veut récupérer.



Attention, **label** est de type `GtkLabel`. Comme pour **container**, nous allons utiliser la macro `GTK_LABEL()`. De plus, faites bien attention, la fonction renvoie un pointeur **const**.

Cette fonction retourne le texte de notre label (avec son formatage, voir plus bas) dans une variable. Faites attention si vous lisez la doc de GTK, il existe une fonction semblable, mais qui ne prend pas en compte le formatage.

Modifier un label

Sur le principe de la dernière fonction, `gtk_label_set_label()` permet de modifier un label.

Code : C

```
void gtk_label_set_label (GtkLabel *label, const gchar *str);
```

Les paramètres :

- **label** : Le nom du label que l'on veut modifier.
- **str** : Le nouveau texte que l'on veut mettre dans le label.

Les Accents



Il y a un bug avec GTK ! Quand j'écris Je suis un zéro, il coupe le texte au niveau de l'accent, ou m'affiche des caractères bizarres !!!

Bonne remarque ! Les textes accentués ne s'affichent pas bien, et ce n'est pas un bug.

Pango, l'indispensable

Pour afficher un texte, GTK utilise Pango (Pan du grec qui veut dire "tous" et go du japonais qui veut dire "langage": tous les langages). C'est une bibliothèque qui est chargée du rendu des caractères et de l'affichage de texte internationalisé. Pour les spécialistes, cette bibliothèque utilise l'Unicode (pour l'encodage des caractères).

Pourquoi alors votre texte ne s'affiche pas comme vous le souhaitez? La raison est que votre système d'exploitation n'utilise pas forcément le même jeu de caractère que notre bibliothèque. Nous allons très rapidement contourner ce problème en utilisant la fonction ci-dessous qui va convertir notre texte :

Code : C

```
gchar* g_locale_to_utf8 (const gchar *opsysstring, gssize len, gsize *bytes_read, gsize *bytes_written, GError **error);
```

Les paramètres :

- **opsysstring** : Ici, nous mettons le texte que nous voulons convertir.
- **len** : C'est la longueur de notre chaîne de caractère (notre texte). Vous pouvez vous embêter à compter, mais personnellement je laisse GTK calculer cela tout seul, en mettant `-1` comme valeur.

- **bytes_read** : C'est un paramètre utile pour les erreurs, il donne le nombre de bytes qui ont été convertis avec succès.
- **bytes_written** : C'est un paramètre utile aussi pour les erreurs, il donne le nombre de bytes qui ont été écrit dans la nouvelle chaîne de caractères.
- **error** : C'est ici que se trouvent les codes d'erreurs (s'il y en a une ^^). Ils sont :
 - G_CONVERT_ERROR_NO_CONVERSION -> La conversion de cette chaîne de caractère n'est pas supportée.
 - G_CONVERT_ERROR_ILLEGAL_SEQUENCE -> L'ordre des bytes n'est pas compatible.
 - G_CONVERT_ERROR_FAILED -> Conversion échouée par une raison inconnue.
 - G_CONVERT_ERROR_PARTIAL_INPUT -> La chaîne de caractères est incomplète à la fin de l'entrée.
 - G_CONVERT_ERROR_BAD_URI -> L'URI est invalide.
 - G_CONVERT_ERROR_NOT_ABSOLUTE_PATH -> Le nom n'est pas un chemin absolu.

Nous mettons les trois derniers paramètres à NULL pour que ça marche.

Cette fonction converti la chaîne, mais ne la modifie pas : elle alloue de la mémoire pour stocker la conversion et retourne un pointeur vers celle-ci. N'oubliez donc pas de libérer cette chaîne avec `g_free`.

Voir aussi

Si cela vous embête d'avoir à convertir toutes vos chaînes accentuées, il existe une solution toute simple. Voir les [astuces dans les annexes](#)

Changer le style d'un label



C'est génial d'afficher du texte, mais j'aimerais le centrer et le mettre en gras. Tu n'a pas une fonction qui pourrait faire ça?

GTK, n'est pas un traitement de texte... 😊, mais ses créateurs nous ont prévu ce genre de fonction.

L'alignement

Le texte est pour le moment aligné à gauche, mais moi je préférerais qu'il soit centré, pas vous?

Pour modifier l'alignement de notre label, la fonction `gtk_label_set_justify()` est tout à fait appropriée, tout simplement par ce qu'elle ne sait faire que ça. 😊

Code : C

```
void gtk_label_set_justify (GtkLabel *label, GtkJustification
jtype);
```

Les paramètres :

- **label** : Le nom de notre label que l'on veut modifier
- **jtype** : Le type d'alignement que l'on veut appliquer à notre label. Il peut prendre une de ces quatre valeurs :
 - GTK_JUSTIFY_LEFT -> Aligne le texte à gauche (par défaut).
 - GTK_JUSTIFY_CENTER -> Aligne le texte au centre.
 - GTK_JUSTIFY_RIGHT -> Aligne le texte à droite.
 - GTK_JUSTIFY_FILL -> Aligne le texte à droite et à gauche à la fois. Le texte est alors justifié.

Le formatage du texte avec les balises

Pour mettre un texte en forme, Pango utilise des balises (comme pour le Zcode).

Avec les balises "simples" ou "courtes"

Les balises ci-dessous vous permettent de personnaliser un texte simplement.

- `` -> Mets le texte en gras
- `<i>` -> Mets le texte en italique

- `<u>` -> Souligne le texte
- `<s>` -> Barre le texte
- `<sub>` -> Mets le texte en indice
- `<big>` -> Rend la police relativement plus grande (+1)
- `<small>` -> Rend la police relativement plus petite (-1)
- `<tt>` -> Met le texte en télétype

Avec Span

La balise `span` est différente des dernières. Elle a des attributs qui lui permettent de modifier la police, la taille, la couleur des caractères.

Elle s'utilise comme ceci :

```
<span attribut1="valeur" attribut2="valeur">...</span>
```

- **font_family** -> Nom de la police de caractère
- **face** -> C'est un autre attribut qui définit la police
- **size** -> C'est la taille de la police. On peut aussi utiliser 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' ou une valeur numérique.
- **style** -> Définit le style des caractères : 'normal', 'oblique' ou 'italic'
- **weight** -> Définit le ton du gras du caractère : 'ultralight', 'light', 'normal', 'bold', 'ultrabold', 'heavy' ou une valeur numérique.
- **variant** -> Met le texte en petites majuscules (smallcaps) ou en normal (normal, valeur par défaut).
- **stretch** -> définit l'espacement des lettres: 'ultracondensed', 'extracondensed', 'condensed', 'semicondensed', 'normal', 'semiexpanded', 'expanded', 'extraexpanded' ou 'ultraexpanded'.
- **foreground** -> Définit la couleur du texte en valeur hexadécimale
- **background** -> Définit la couleur du fond texte en valeur hexadécimale
- **underline** -> Définit le soulignement du texte: 'single', 'double', 'low' ou 'none'.
- **underline_color** -> Définit la couleur du soulignement en valeur hexadécimale
- **rise** -> Définit l'élévation du texte (en indice ou exposant) en valeur décimal (les valeurs négatives sont possibles, pour mettre notamment notre texte en indice)
- **strikethrough** -> Pour barrer son texte. La valeur doit être soit TRUE ou FALSE
- **strikethrough_color** -> Définit la couleur de la ligne qui barre le texte en valeur hexadécimale
- **fallback** -> Si votre caractère n'est pas disponible dans le police choisie, alors une police qui contient ce caractère sera choisi. Elle est activée par défaut.
- **lang** -> Définit la langue du texte

Maintenant, il y a une chose primordiale à ne pas oublier: il faut dire que l'on utilise les balises de pango.

Pour cela nous avons deux choix, soit on intègre ces balises directement dans notre fonction de conversion, soit nous utiliserons une fonction spécifique.

On intègre nos balises à notre fonction de conversion

Cette fonction très pratique permet de convertir à la fois le texte et les balises. Il vous suffit d'écrire vos balises avec votre texte dans `g_locale_to_utf8` puis de rajouter la fonction ci-dessous après la conversion :

Code : C

```
void gtk_label_set_use_markup (GtkLabel *label, gboolean setting);
```

Les paramètres :

- **label** : Le label à qui on veut appliquer les balises.
- **setting** : On met TRUE pour que ça marche.

On utilise pas notre fonction de conversion

Si vous n'avez pas de caractères spéciaux à convertir, l'utilisation de cette fonction sera plus adaptée.

Code : C

```
void gtk_label_set_markup (GtkLabel *label, const gchar *str);
```

Les paramètres :

- **label** : Le label à qui on veut appliquer les balises.
- **str** : On met ici le texte avec les balises que l'on souhaite afficher

C'est fini, il ne reste plus qu'à afficher le label.

Exemple

Je vous propose un petit exemple qui reprend les fonctions vues dans ce chapitre ! 😊

Code : C

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
    GtkWidget* Fenetre = NULL;
    GtkWidget* Label = NULL;
    gchar* TexteConverti = NULL;

    gtk_init(&argc, &argv);

    Fenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL); // Définition de
la fenêtre
    gtk_window_set_title(GTK_WINDOW(Fenetre), "Le texte avec les
labels"); // Titre de la fenêtre
    gtk_window_set_default_size(GTK_WINDOW(Fenetre), 300, 100); //
Taille de la fenêtre

    TexteConverti = g_locale_to_utf8("<span face=\"Verdana\"
foreground=\"\#73b5ff\" size=\"xx-large\"><b>Le site du
Zéro</b></span>\n <span face=\"Verdana\" foreground=\"\#39b500\"
size=\"x-large\">Le tuto GTK</span>\n", -1, NULL, NULL, NULL);
//Conversion du texte avec les balises
    Label=gtk_label_new(TexteConverti); // Application de la
conversion à notre label
    g_free(TexteConverti); // Libération de la mémoire

    gtk_label_set_use_markup(GTK_LABEL(Label), TRUE); // On dit que
l'on utilise les balises pango

    gtk_label_set_justify(GTK_LABEL(Label), GTK_JUSTIFY_CENTER); //
On centre notre texte

    gtk_container_add(GTK_CONTAINER(Fenetre), Label); // On ajoute
le label a l'interieur de 'Fenetre'

    gtk_widget_show_all(Fenetre); // On affiche 'Fenetre' et tout ce
qu'il contient

    g_signal_connect(G_OBJECT(Fenetre), "delete-event",
G_CALLBACK(gtk_main_quit), NULL); // Je ne commente pas cette
```

```
fonction, vous la verrez dans le chapitre suivant.  
  
    gtk_main();  
  
    return EXIT_SUCCESS;  
}
```

Et voici le résultat :



Maintenant c'est à vous de bosser... 😊

Ce chapitre est fondamental (comme tout les chapitre de cette partie 🤪), vous allez vous servir des labels partout où vous voudrez afficher du texte.

Dans le chapitre suivant, vous allez découvrir le fonctionnement des événements. Là aussi, ce chapitre est primordial. Je le qualifierais même comme le chapitre le plus important. Sans lui, vous ne saurez rien faire avec GTK. Si si, c'est vrai !

By [Picxime](#)

Gestion des évènements : les signaux

Nous allons maintenant voir comment intercepter les actions de l'utilisateur et agir en conséquence. Au programme, explications, exemples et exercices.

Comment ça marche ?

Nous allons traiter ici ce qu'on appelle la programmation événementielle ! Cela consiste à intercepter les actions de l'utilisateur pour les traiter : clic sur un bouton, validation d'une zone de texte, etc.. Sans ça, votre programme ne fera rien ! En fait, c'est cette gestion qui fera le "gros" du programme, tous se passe dedans !

Principe

Vos programmes ne suivront désormais plus du tout le même plan. Je m'explique : 😊

- Avant, vos programmes étaient dis **linéaires** : votre code s'exécutait dans l'ordre où vous l'aviez écrit (du haut vers le bas), vous étiez d'ailleurs assez limités par cette contrainte. De plus, l'utilisateur n'avait aucune liberté, il ne pouvait rien faire d'autre que ce que vous lui proposiez.
- Maintenant, c'est une toute autre façon de penser. Dans votre programme, vous définissez votre interface, les boutons et tout ça, on lance le tout, puis on passe la main à l'utilisateur. C'est lui qui choisit ce qu'il fait, quand, et comment.

Cette méthode de programmation est dite **événementielle**, elle suit le principe **action -> réaction**.

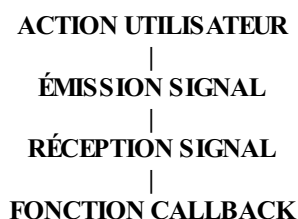
Le principe est vraiment simple : lorsqu'un évènement a lieu avec un widget, celui ci émet ce qu'on appelle un signal (celui-ci sera différent pour chaque action que le widget gère), l'objectif du jour est de savoir faire réagir notre application à la réception de ces signaux.

GTK s'occupe de tout, donc la partie technique ne nous regarde pas ; cependant il faut bien comprendre comment ça fonctionne.

Il n'y a qu'une seule chose à faire. On va utiliser une fonction qui dira à GTK :

- De quel widget on veut traiter un évènement.
- Le signal (l'évènement) que l'on veut traiter.
- Le nom d'une fonction (faite par vous), à appeler lorsque cet évènement survient. Ce type de fonction est appelé **fonction callback**.

En gros, on peut résumer avec un petit schéma :



Passons maintenant à la pratique ! Voilà le code de base sur lequel nous allons travailler. Créez un projet et collez ce code dedans :

Code : C

```

#include <stdlib.h>
#include <gtk/gtk.h>

void mafonction(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}

int main(int argc, char **argv)
{
    /* Variables */
  
```

```

GtkWidget * MainWindow = NULL;

/* Initialisation de GTK+ */
gtk_init(&argc, &argv);

/* Création de la fenêtre */
MainWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
g_signal_connect(G_OBJECT(MainWindow), "delete-event",
G_CALLBACK( mafonction ), NULL);

/* Affichage et boucle événementielle */
gtk_widget_show(MainWindow);
gtk_main();

/* Fermeture de GTK+ */
gtk_exit(EXIT_SUCCESS);
return EXIT_SUCCESS;
}

```

Observons ce code, et essayons de le comprendre. La seule nouveauté est la petite fonction en haut, normalement vous devriez vous dire :



À quoi sert-elle ?

Quand est-elle appelée ? Comment ?

Qu'est-ce qu'on doit mettre dedans ? Etc ..

À quoi sert la fonction `g_signal_connect` au milieu du code ?

Autant de questions auxquelles je vais tenter de répondre.

Gérer une action

Penchons-nous déjà sur cette ligne :

Code : C

```

g_signal_connect(G_OBJECT(MainWindow), "delete-event", G_CALLBACK (
mafonction ), NULL);

```

L'appel de cette fonction permet de gérer un évènement. Voyons les paramètres :

- `MainWindow` : Ce premier paramètre doit être le widget dont on veut gérer un évènement, en le mettant dans la macro `G_OBJECT`. Ici, on gère un évènement de la fenêtre.
- `"delete-event"` : On écrit le nom du signal que l'on souhaite intercepter dans une chaîne de caractères. Il existe un nom de signal pour chaque signal existant. En l'occurrence, `"delete-event"` est émis lorsque on clique sur la croix d'une fenêtre.
- `mafonction` : Il faut mettre ici le nom d'une **fonction callback** qui sera appelée lors de l'évènement, en le mettant dans la macro `G_CALLBACK`.
- `NULL` : Pour finir, on donne un paramètre passé automatiquement à la fonction donnée juste avant. On s'en sert pour passer des données à la fonction. Ici, on met `NULL`.

Sachant que le prototype est :

Code : C

```

gulong g_signal_connect(gpointer *object, const gchar *name,
GCallback func, gpointer func_data );

```

On peut dire que : Quand **object** émet le signal **name**, GTK+ appelle la fonction **func** avec les paramètres **object** et **data**. Donc à ce moment là, GTK génère cet appel et l'exécute :

Code : C

```
func(object, data);
```

La fonction en question devra toujours avoir le prototype suivant :

Code : C

```
void mafonction(GtkWidget *widget, gpointer data);
```



En fait, ce prototype pourra changer pour certains évènements, mais ce sont des cas particuliers. Nous en parlerons en temps voulu.

Ainsi, on a accès au widget qui a émis le signal par le premier paramètre, et à une donnée quelconque par le deuxième. Maintenant, reprenez votre code, vous êtes maintenant en mesure de le comprendre : le programme est quitté lorsqu'on clique sur la croix de la fenêtre.

La boucle principale

Pour gérer les évènements, GTK+ utilise une **boucle évènementielle**. C'est une boucle *infinie* (enfin presque, faut bien qu'on puisse quitter) dans laquelle GTK+ s'occupe d'émettre les signaux, et d'appeler les fonctions callback nécessaires. Pour lancer cette boucle (le plus souvent, pour "lancer le programme" en quelque sorte), on appelle la fonction `gtk_main` :

Code : C

```
void gtk_main(void);
```

Pour quitter cette boucle, la fonction est :

Code : C

```
void gtk_main_quit(void);
```

Vous savez maintenant tout ce qu'il faut savoir pour bien aborder la gestion des évènements. Assurez-vous d'avoir bien compris, je vous conseille de relire l'article plusieurs fois si besoin. Je sais que tout cela peut vous paraître un peu flou, mais après tout c'est quand même plus simple que les pointeurs ! 🤪 (à moins que..?)

Exercices

Maintenant que vous avez (plus ou moins) compris comment on traitait les événements, voilà deux exercices. Il serait vraiment essentiel de les réussir, et pour vous y aider je ne donne pas de solution : l'important, c'est que ça marche !

Pour bien commencer, prenez le code de la partie précédente. N'oubliez pas que tout se passe dans la fonction qui est en haut !



Exercice 1

Faites un programme qui fait ceci lorsque on clique sur la croix :

- Ecrire le titre de la fenêtre dans la console, *en inversant l'ordre des lettres* ! 😊
- Quitter le programme.

Assez facile, pensez à activer la console si elle n'apparaît pas ! (Projet > Options)

Exercice 2

Pas beaucoup plus dur, mais plus marrant ! 😊

- Ecrivez un programme qui ouvre une deuxième fenêtre quand on clique sur la croix de la première.
- Quitter le programme quand on clique sur la croix de la deuxième.
- La deuxième fenêtre contiendra le titre de la première dans un label.

Attention, ici on a deux signaux à gérer, un pour chaque fenêtre !

Exercice 3

Vous devriez y arriver, mais il est plus difficile.

- Créez une fenêtre, avec la première lettre de votre prénom dedans (dans un label).
- Quand on clique sur la croix de la fenêtre, on change la lettre affichée par la suivante.
- Quand on est arrivé à la fin du nom, on quitte.

Voilà enfin cette première partie sur GTK+ terminée, vous avez normalement acquis les bases et êtes prêts à faire des choses plus intéressantes ! Au programme de la seconde partie : du concret ! Découvertes des widgets les plus utilisés, au moins un TP, bref : de quoi vous occuper. 😊

Partie 2 : Annexes

Quelques bonus, trucs et astuces dont certains inédits rien que pour vous ! 😊

Trucs et astuces divers

Ce chapitre contient diverses astuces qui pourraient bien vous servir, et ce quel que soit votre niveau/avancement.

N'hésitez pas à venir jeter un coup d'oeil de temps en temps ! 😊

Quelques Macros...

Les conventions de GTK disent que chaque widget doit posséder trois macros (minimum) permettant de manipuler les Widgets :

- **GTK_WIDGET(widget)** : Permet de *caster* (transformer) un widget en son type réel.
Par exemple, pour modifier les paramètres d'une fenêtre, vous devez passer aux fonctions un `GtkWindow` et non un `GtkWidget`, ce qui se fait donc avec la macro `GTK_WINDOW`.
- **GTK_WIDGET_CLASS(widget)** : Même chose, sauf qu'on obtient un `GtkWidgetClass*`.
Utile lors du développement de widgets, mais pas (ou très peu) utilisé par l'utilisateur (vous).
- **GTK_IS_WIDGET(widget)** : Permet de savoir si un widget est de type `GtkWidget`.
Renvoie "vrai" si le widget est bien du type précisé.

* *WIDGET* : Quand Widget est écrit ainsi, cela signifie que le mot est à remplacer par le type voulu.

Ex : pour les `GtkWindow`, les macros sont :

Code : C

```
#define GTK_WINDOW ...  
#define GTK_WINDOW_CLASS ...  
#define GTK_IS_WINDOW ...
```

Rien de bien passionnant je vous l'avoue, mais vous devez connaître l'existence de ces macros, ou tout au moins la première !



Astuce par [Guimers8](#)

Retour à la console

Apprenez chers amis, que lors du développement d'une vraie application GTK, ou même d'un simple petit programme, la console peut se révéler utile. Je vous conseille très fortement d'activer la console pendant le développement (on l'enlève pour la distribution 😊), car elle permet d'afficher différentes choses telles que :

- En cas d'erreur d'installation d'un composant, mais qui ne gêne pas la compilation, vous pouvez avoir des erreurs qui apparaissent.
- En cas d'erreur d'utilisation des certains widgets, vous pouvez voir des *Warnings*. Très utile !
- Pour faire du debug, par exemple pour afficher simplement un retour de fonction (pour faire des tests).

Ces trois raisons sont largement suffisantes ! Quand vous avez un problème, ne posez pas de questions sur le forum avant d'avoir vérifié qu'il n'y a pas d'erreurs !

Astuce par [Guimers8](#)

Problèmes d'encodage UTF-8

GTK+ utilise l'encodage UTF-8 pour le traitement et l'affichage des chaînes de caractères, ce qui peut poser problème et gêner l'affichage des caractères spéciaux et des accents. On aura alors une erreur de ce type dans la console :

Code : Console

```
** (programme.exe) : WARNING **: Invalid UTF8 string passed to pango_layout_set_text
```

La solution qui permet de remédier à ce problème est simple, il suffit de convertir la chaîne en question en UTF-8, comme vous l'avez vu dans le chapitre sur les labels :

Code : C

```
#define UTF8(string) g_locale_to_utf8(string, -1, NULL, NULL, NULL)

/* encodage */
gchar *chaîne = UTF8("Un texte accentué, avec des caractères
bizarres... $23, 32?, £18...");
/* utilisation de la chaîne */
<...>
/* libération de la chaîne allouée */
g_free(chaîne);
```

Cependant, il est tout de même assez lourd de devoir passer par un encodage, puis de libérer la mémoire après utilisation... Alors qu'il y a bien plus simple !

Pourquoi ce problème ?

Nous avons un conflit d'encodage entre les chaînes traitées par GTK+ (UTF-8), et celles que vous tapez dans votre IDE (ISO-38547 ou autre...). Conclusion : votre éditeur n'utilise pas le même encodage que GTK+.

Comment le résoudre ?

Pour résoudre le problème, il faudrait que votre éditeur, et donc vos fichiers sources, utilisent l'encodage UTF-8. Voici comment faire :

Sous Linux : il est très probable que vos fichiers soient déjà encodés en UTF8, ainsi pas de problème. Si ce n'est pas le cas, vous pouvez généralement régler cela dans les préférences de votre éditeur. Pour info, Code::Blocks utilise l'UTF-8 par défaut.

Sous Windows : sur cet OS, le problème est systématique. Soit vous choisissez d'utiliser un éditeur externe dont on peut choisir l'encodage, soit vous utilisez les [nightly builds de Code::Blocks](#).

Sous Mac OS : XCode vous permet de choisir l'encodage (menu *Format > File encoding > Unicode UTF8*). Si vous utilisez un éditeur externe, comme [Smultron](#), vous pourrez également utiliser l'encodage de votre choix. 😊

C'est tout ?

Oui. 😊 Réenregistrez vos fichiers avec le nouvel encodage, puis compilez : comme par magie, plus besoin de convertir vos chaînes !
Merci qui ?

Astuce par [Guimers8](#)

Texte accentué

Dans le [chapitre sur les labels](#), vous avez appris que GTK utilisait l'encodage UTF-8, mais que ce n'était pas forcément votre encodage local et qu'il fallait donc convertir les chaînes accentuées. Cela ne concerne pas que les labels mais tout les textes que GTK est susceptible d'afficher. (titre de fenêtre, boutons, etc...)

Vous découvrez donc une *petite* fonction permettant de convertir une chaîne en UTF-8 :

Code : C

```
gchar* g_locale_to_utf8 (const gchar *opsysstring, gssize len, gsize
```

```
*bytes_read, gsize *bytes_written, GError **error);
```

La plupart du temps, on ne se sert que du premier paramètre; et comme cette fonction est très longue, je vous propose d'écrire une petite macro qui fera la conversion mais qui prendra moins de place. La voilà :

Code : C

```
#define UTF8(string) g_locale_to_utf8(string, -1, NULL, NULL, NULL)
```

Ca sera plus pratique à utiliser comme ça ! Cependant, n'oubliez pas de libérer la mémoire lorsque c'est nécessaire ! 😊

C'est une petite astuce toute bête, que vous pouvez appliquer pour d'autres fonctions *longues* ! 😊

Astuce par [Guimers8](#)

Il n'y a pas de QCM, cela ne servirait à rien, vous n'avez réellement rien appris ! 😊

Puissent ces astuces vous servir !

Le cours n'est bien entendu pas encore terminé, les chapitres sortiront au fur et à mesure. Sachez que le plan du cours est (en gros) bien défini et que nous aborderons quasiment tous les points pouvant vous intéresser, répartis sur au moins quatre parties dont une consacrée aux annexes. Rassurez-vous, **le programme est plus que complet** ! 😊

Si vous avez des questions ou suggestions relatifs à un chapitre en particulier, envoyez un MP à l'auteur du chapitre en question (en bas de page), pas à moi !

Pour les problèmes de code, n'oubliez pas que le forum est là.

Picxime, Guimers8 & Im@GinE remercient les zCorrecteurs Petrus6, ptipilou et Ziamé qui ont participé à la correction ainsi que la validatrice Faeria MC (Morphée) qui s'occupe de notre tutorial.

Citation

[Important] Le Tutoriel GTK+ est actuellement en pause !

La rédaction de ce tutoriel a commencé il y a un peu plus d'un an, nous étions tous très motivés et avons travaillé dur pendant à peu près 9 mois. Mais depuis le début de l'année, l'acharnement au travail a diminué... Ainsi depuis plusieurs mois certains attendent les nouveaux chapitres, nous avons en effet écrit en partie les chapitres, mais ils ne sont plus les mêmes, trop imparfaits, et nous avons moins la motivation (et aussi surtout le temps ! 😊) de les terminer correctement.

Nous avons ainsi décidé de mettre en pause la réalisation du tutorial car il devient important pour tous les 3 de consacrer plus de temps au travail scolaire.

Souhaitant que vous puissiez continuer d'apprendre GTK+, voici une liste de divers Tutos disponibles sur le net :

- [\[Cours GTK\]](#) : Ce cours développera en détail la majorité des fonctions de GTK+ tout en fournissant des exemples concrets. De ce fait, ce cours sera une sorte de tutorial couplé à un manuel de référence complet.
- [\[Gtk.org\]](#) : Le tutoriel officiel (donc en anglais !), par les créateurs de GTK+. Il est toujours intéressant d'étudier un document édité par les codeurs de la bibliothèque.
- [\[Gtk par l'exemple\]](#) : Ce tutoriel a pour but de vous guider dans la réalisation d'une interface graphique en C grâce à GTK+ au travers l'exemple de la réalisation d'un éditeur de texte. Très instructif pour la pratique de GTK 😊.
- [\[Tout sur GTK\]](#) : Cette page recense un grand nombre de contenus liés à GTK (documents, tutos, forum,...)
- [\[Gtk-Fr\]](#) : Le tutoriel du wiki francophone dédié à GTK. Malgré quelques (grosses ? 😊) lacunes pourtant essentielles, ce tutoriel est tout de même intéressant, avec de la bonne volonté biensûr.

Et en dernier cadeau, quelques documents sélectionnés (en anglais) :

- [\[Spécial GtkTextView\]](#) : Un document de plusieurs pages, à étudier absolument si vous travaillez avec les GtkTextView. Recherche, coloration, insertion d'images,... tout y est. A combiner avec le tuto [\[Gtk par l'exemple\]](#), cité plus haut.
- [\[Créer un widget\]](#) : Comment faire son propre widget, [ceci](#) peut aussi être utile. Et [ça aussi](#) d'ailleurs.
- [\[Documentation\]](#) : La documentation officielle 😊.

Merci pour tous les messages de soutien que vous avez pu nous envoyer et à tout ce qui nous ont aidé ! 😊
Bon codage ! 😊