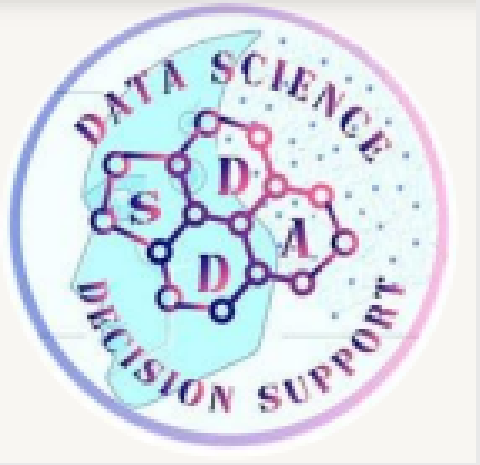


Les interfaces graphiques avec gtkmm



<https://sdadclub.tech/>



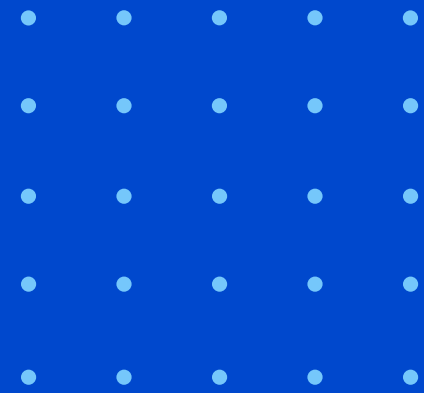
Ce cours est réalisé par :

ASAKOUR Ihsane

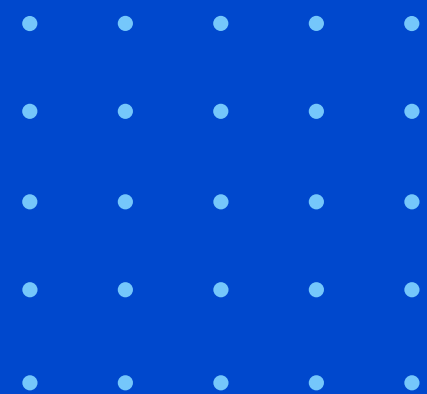
étudiante en Sciences des
données et aide à la décision à
l'université Cadi Ayyad et une
membre du club SDAD.



ihane.asakour20@gmail.com

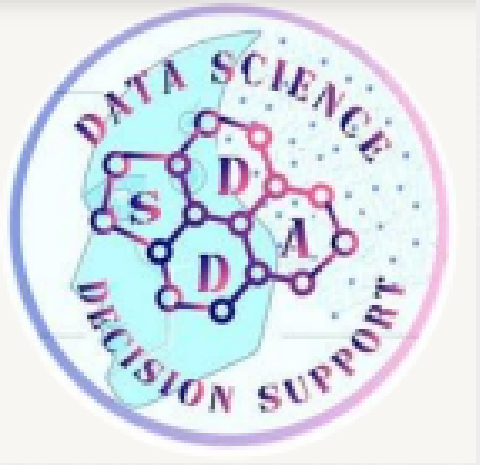


Plan :



- Introduction
- Les Widgets
- L'Héritage
- Les Conteneurs
- Les signaux et les fonctions de rappel
- Les Références:





Introduction :



Présentation de GTK+ :

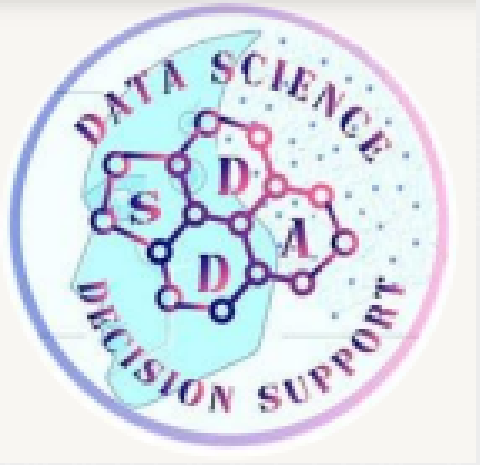
GTK+ est un ensemble de bibliothèques permettant de créer des interfaces graphiques, donc des applications fenêtrées, en C.

GTK+ est libre, ce qui vous permet de créer des logiciels propriétaires sans contrainte.

GTK+ est portable ; il fonctionne sur la plupart des systèmes d'exploitation :

Elle est utilisable dans de nombreux autres langages tels C++, PHP, Java et Python pour ne citer que ceux-ci.





Présentation de GTKmm :

Gtkmm est une surcouche de GTK+ pour le langage C++.

Elle permet de créer des interfaces graphiques en utilisant les mécanismes du C++ plutôt que ceux du C.

Elle utilise, bien sûr, la programmation orientée objet (héritage, polymorphisme, etc.).

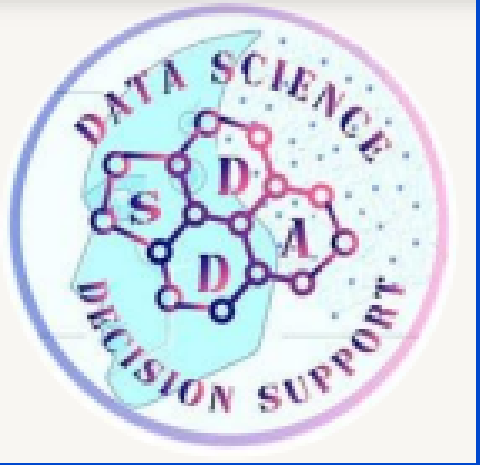
Il existe également les surcouches des autres bibliothèques présentées plus haut:

glibmm

libglademmm

etc



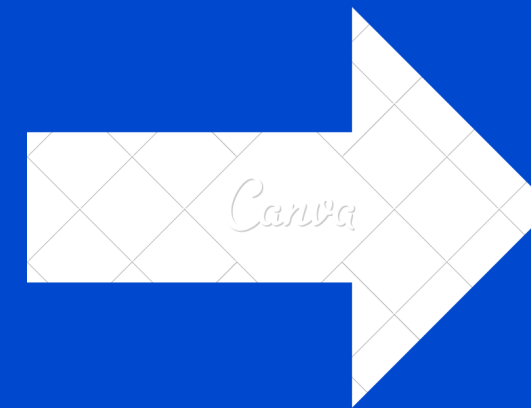


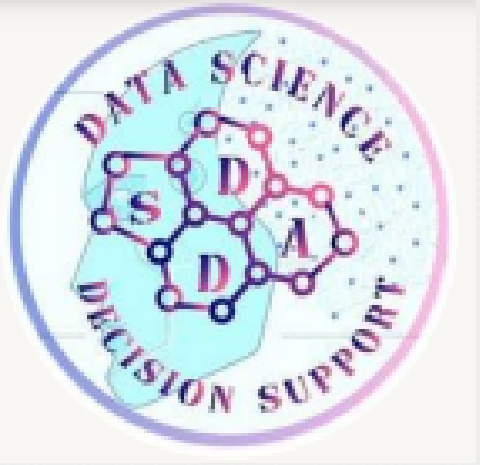
Les Widgets :

Première fenêtre :

```
#include <gtkmm/main.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);
    Gtk::Window fenetre;
    Gtk::Main::run(fenetre);
    return 0;
}
```





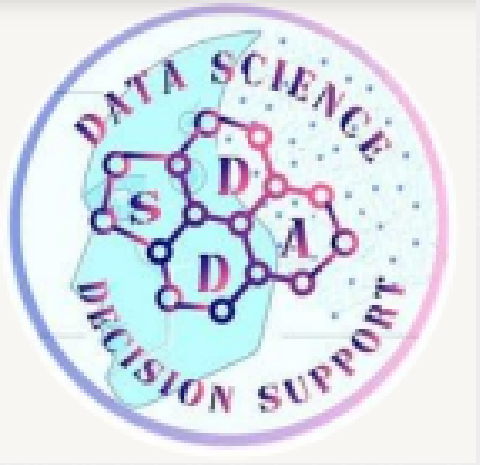
Vous pouvez remplacer **#include <gtkmm/main.h>** et **#include <gtkmm/window.h>** par **#include <gtkmm.h>** .mais de cette façon, nous incluons toutes les classes de gtkmm, pas seulement celles nécessaires

Gtk::Main app(argc, argv): initialiser gtkmm,le passage des arguments permet d'éviter le plantage du programme

Gtk::Window fenetre : permet la création du fenêtre

Gtk::Main::run(fenetre) : afficher la fenêtre et entre dans la boucle principale (**Pour que l'application s'exécute en continu, il doit y avoir une boucle infinie**)





Modifier un widgets :

fenetre.set_title("Ma belle fenêtre !") : Modifier le titre de votre fenêtre

std::string titre = fenetre.get_title() : Obtenir le titre de votre fenêtre

fenetre.set_icon_from_file("icone.png") : Modifier l'icône de votre fenêtre



fenetre.maximize() : Maximiser la taille de la fenêtre pour occupe la taille de l'écran.

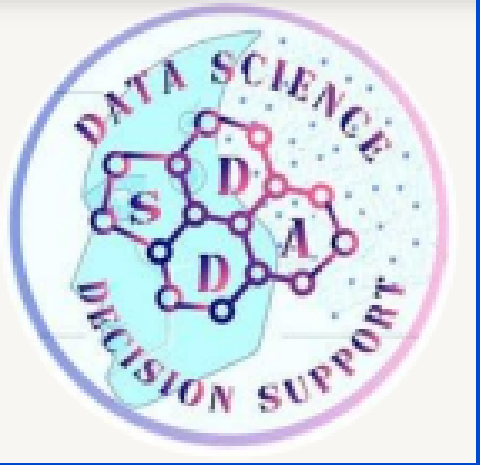
fenetre.unmaximize() : L'opération inverse lui rend sa taille d'avant

fenetre.fullscreen() : Pour que la fenêtre soit en mode plein écran

fenetre.unfullscreen() : Elle redevienne comme avant

fenetre.set_position(Gtk::WIN_POS_CENTER) : Déplacer la fenêtre au centre de l'écran





Ajout d'un widget à la fenêtre :

Exemple : Bouton

```
#include <gtkmm/button.h>
#include <gtkmm/main.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv); //Initialise gtkmm et regarde dans les arguments du programme.

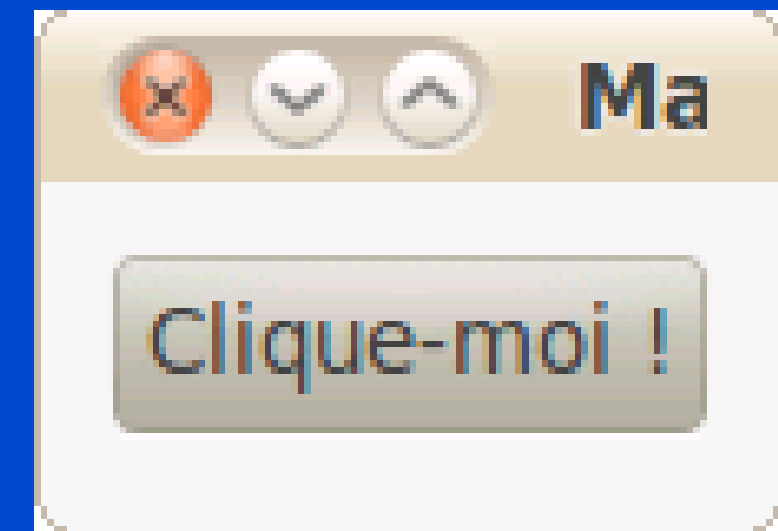
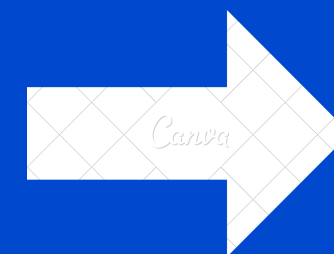
    Gtk::Window fenetre; //Création de la fenêtre.
    fenetre.set_title("Ma belle fenêtre !"); //Modifier le titre de la fenêtre.
    fenetre.set_border_width(10); //Ajouter une bordure (invisible) de 10px à la fenêtre.

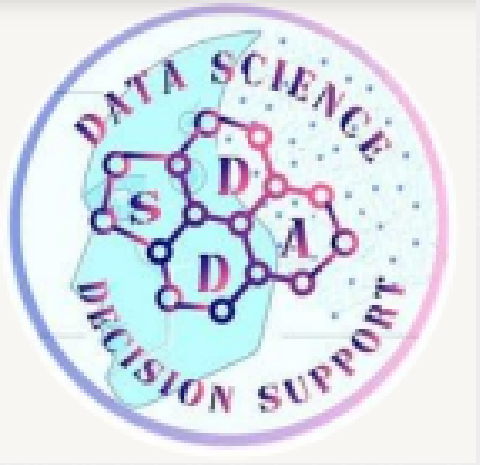
    Gtk::Button bouton("Clique-moi !"); //Création d'un bouton.
    fenetre.add(bouton); //Ajout du bouton à la fenêtre.

    bouton.show(); //Afficher le bouton.

    //Affiche la fenêtre et démarre la boucle, qui se terminera lorsqu'on fermera le programme.
    Gtk::Main::run(fenetre);

    return 0;
}
```





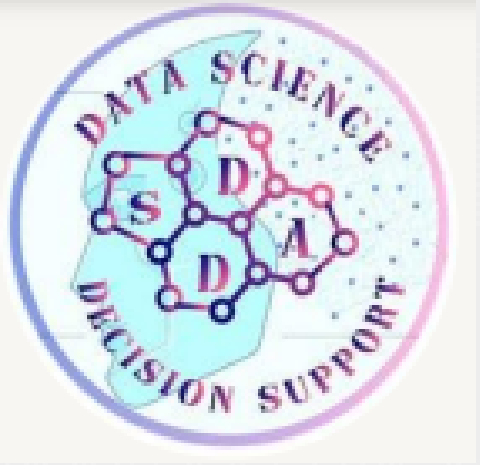
Les Stock Items :

Les Stock Items sont très utiles pour développer des applications sans aller chercher des icônes un peu partout à travers le Net.

En effet, grâce à eux, nous pouvons créer des boutons, mais aussi des éléments dans un menu et dans une barre d'outils, avec une étiquette (son texte) et une icône prédéfinies

Cela signifie que vous codez d'une certaine façon et vous n'avez pas besoin de fournir les icônes avec l'exécutable.





Exemple :

```
#include <gtkmm/button.h>
#include <gtkmm/main.h>
#include <gtkmm/stock.h> //Ne pas oublier d'inclure ce fichier pour pouvoir utiliser les Stock Items.
#include <gtkmm/window.h>

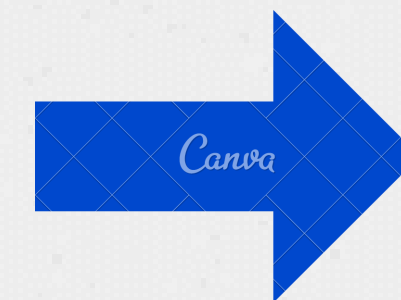
int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);

    Gtk::Window fenetre; //Création de la fenêtre.

    Gtk::Button bouton(Gtk::Stock::CANCEL); //Création d'un bouton à partir d'un Stock Item.
    fenetre.add(bouton); //Ajout du bouton à la fenêtre.
    bouton.set_can_focus(false); //Empêcher le bouton d'avoir le focus.
    bouton.show(); //Afficher le bouton.

    Gtk::Main::run(fenetre);

    return 0;
}
```



Il ne faut pas oublier d'ajouter la ligne **#include <gtkmm/stock.h>**





L'héritage :

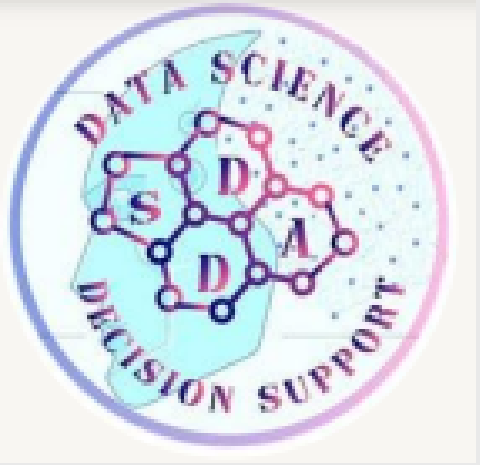
Nous pouvons utiliser l'héritage pour créer de nouveaux widgets. Ces widgets dériveront des widgets existants.

Pour un vrai projet, nous allons créer une classe pour la fenêtre principale.

C'est dans celle-ci que nous modifierons son titre, son icône et que nous y ajouterons des widgets.

Cela permet de mieux organiser le code.





Le fichier main.cpp :

Ce fichier restera toujours identique :

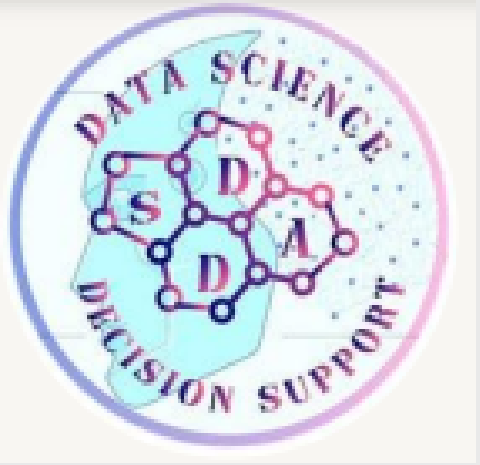
```
#include <gtkmm/main.h>

#include "Fenetre.hpp"

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);
    Fenetre fenetre;
    Gtk::Main::run(fenetre);
    return 0;
}
```

Comme vous pouvez le constater, nous utilisons une classe Fenetre que nous allons créer à l'instant.





Le fichier Fenetre.hpp :

C'est dans ce fichier que nous allons créer la classe Fenetre :

```
#ifndef DEF_FENETRE
#define DEF_FENETRE

#include <gtkmm/window.h>

class Fenetre : public Gtk::Window {
    public :
        Fenetre();
};

#endif
```

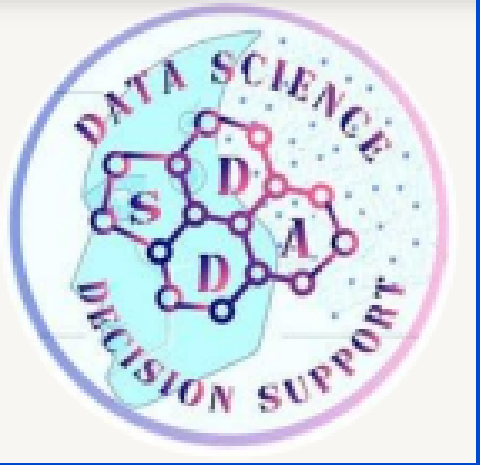
Premièrement, nous écrivons toujours la protection avec **#ifndef**, **#define** et **#endif**.

Ensuite, nous incluons seulement le nécessaire

Ensuite, nous créons notre classe Fenetre, qui hérite de **Gtk::Window**, de façon public.

Ensuite, il y a le constructeur. vous devriez en avoir l'habitude.



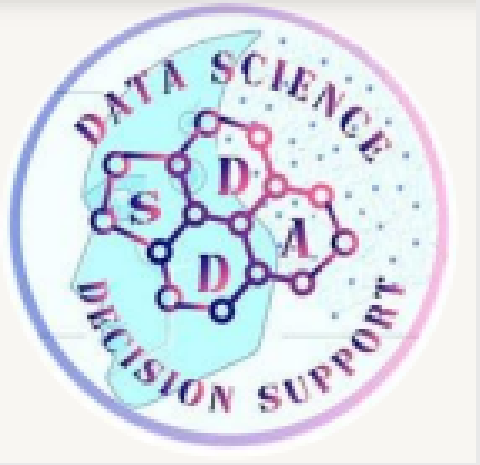


Le fichier Fenetre.cpp :

Dans ce fichier, Nous allons modifier notre fenêtre

```
#include "Fenetre.hpp"

Fenetre::Fenetre() {
    //Modification de la fenêtre.
    set_title("Ma fenêtre héritée !");
    //Autres modifications de la fenêtre.
}
```



Les Conteneurs :



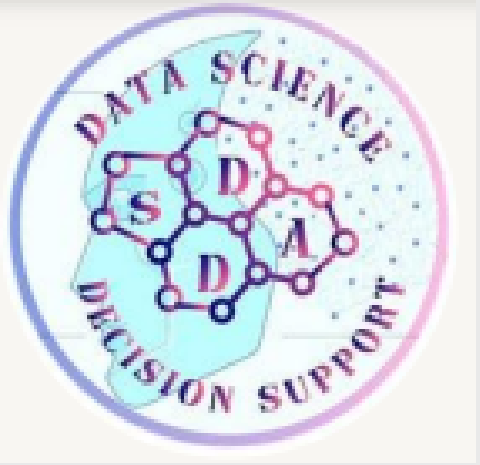
Les conteneurs permettent d'ajouter un ou plusieurs widgets dans une fenêtre.
Donc, c'est une façon d'organiser des widgets.

Certains conteneurs ne peuvent contenir qu'un seul widget.

D'autres, ceux que nous utiliserons la plupart du temps, permettent d'en contenir
plusieurs

Cela nous permet d'organiser la façon dont les widgets sont disposés dans la fenêtre.





Les boîtes :

Les boîtes permettent d'aligner des widgets soit horizontalement, soit verticalement.

Pour utiliser les boîtes, vous devez inclure le fichier approprié **#include <gtkmm/box.h>**





Les boîtes verticales :

Il est possible d'en créer une comme ceci : **Gtk::VBox** `boiteV(false, 10);`

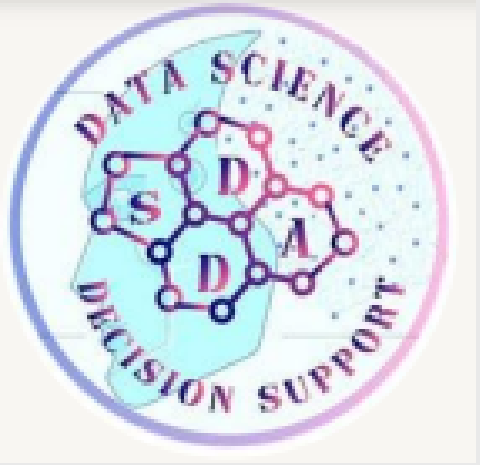
1er paramètre indique si oui ou non les widgets prendront le même espace

2ème paramètre indique la taille minimale (une marge) entre chaque widget

Les boîtes horizontales :

Ces boîtes se créent et s'utilisent de la même manière que les boîte verticales, donc je ne vais pas m'étendre sur leur fonctionnement. Ce sont les **Gtk::HBox**



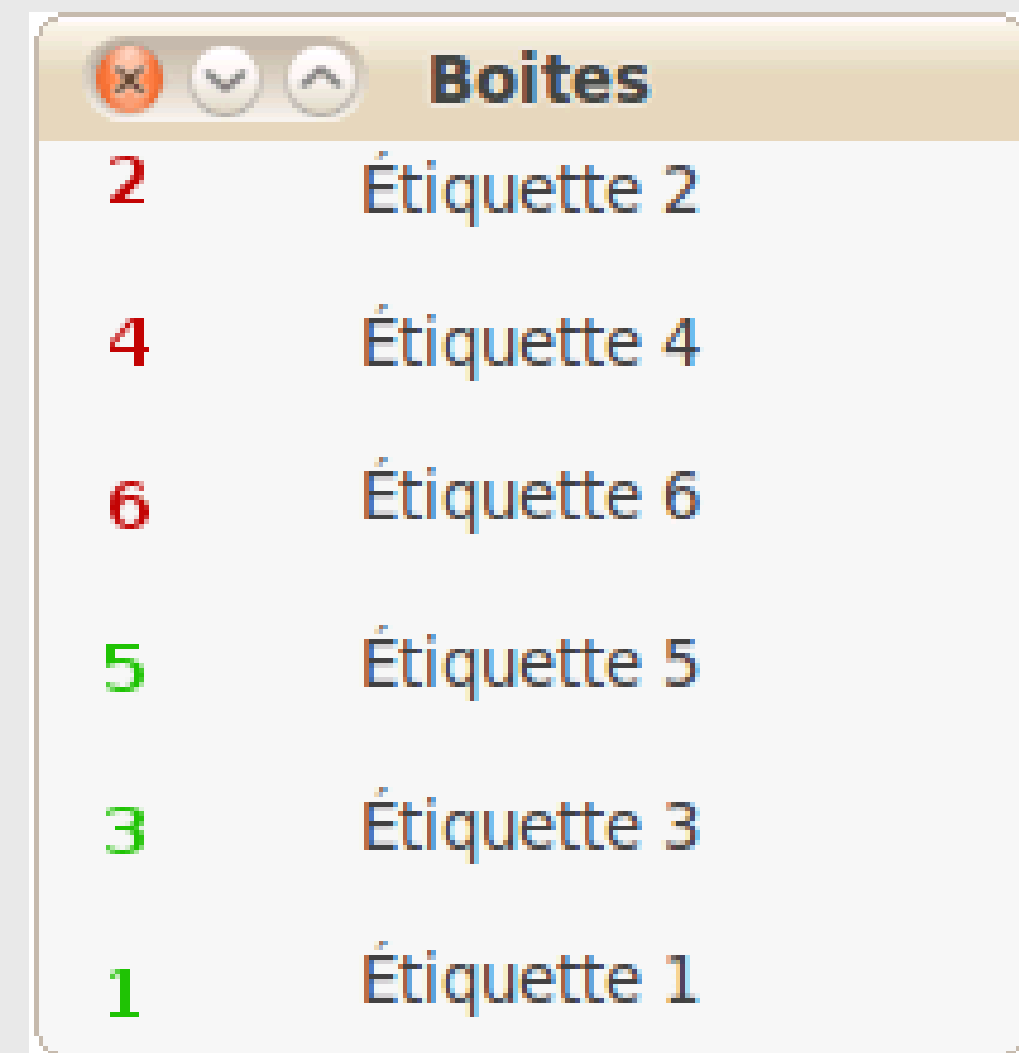
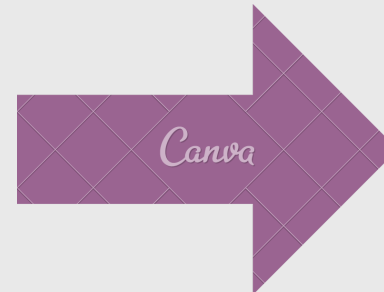


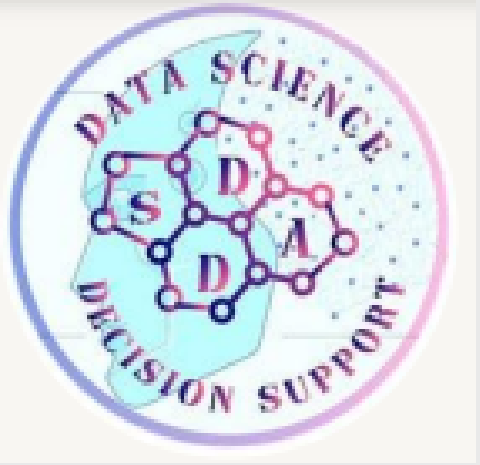
Ajout de *widgets* dans une boîte :

Il y a deux méthodes pour ajouter un widget dans une boîte. **pack_start()** et **pack_end()**

```
Gtk::Label etiquette1("Étiquette 1");  
Gtk::Label etiquette2("Étiquette 2");  
Gtk::Label etiquette3("Étiquette 3");  
Gtk::Label etiquette4("Étiquette 4");  
Gtk::Label etiquette5("Étiquette 5");  
Gtk::Label etiquette6("Étiquette 6");
```

```
boiteV.pack_end(etiquette1); 1  
boiteV.pack_start (etiquette2); 2  
boiteV.pack_end(etiquette3); 3  
boiteV.pack_start (etiquette4); 4  
boiteV.pack_end(etiquette5); 5  
boiteV.pack_start (etiquette6); 6
```





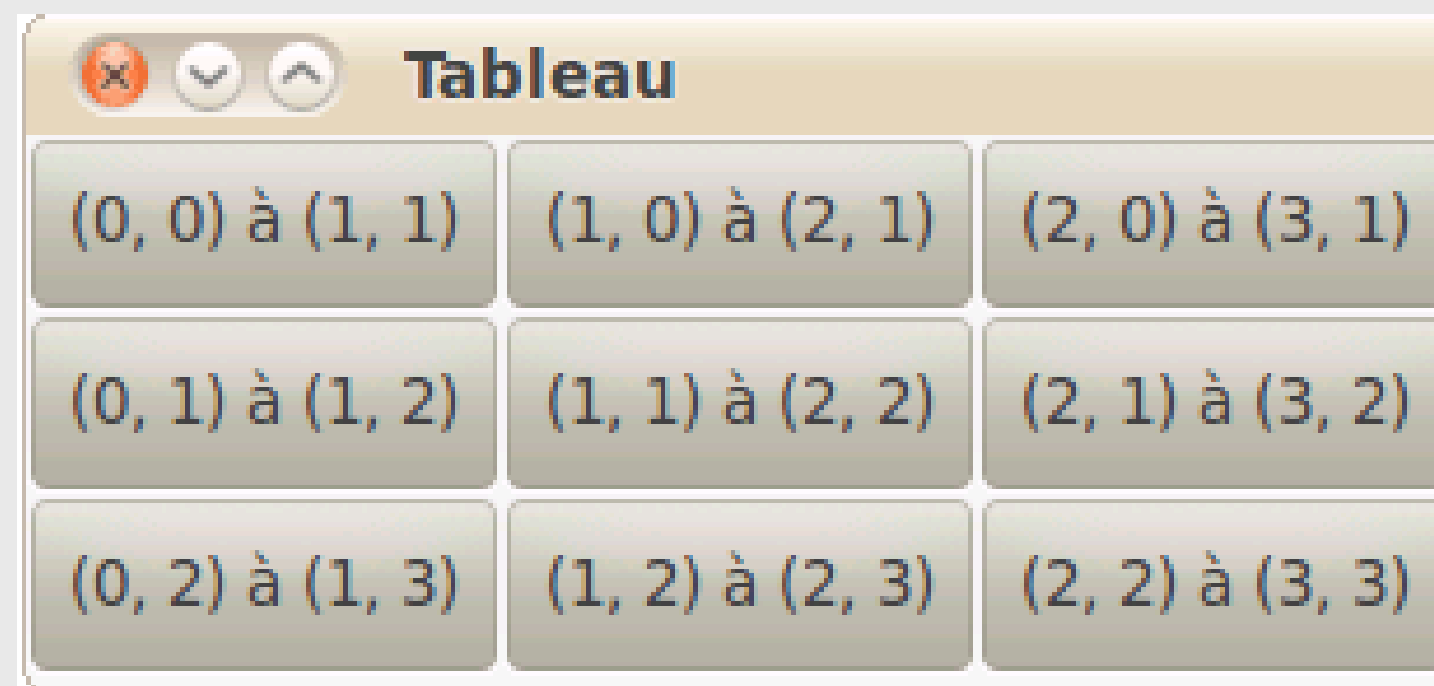
Les Tableaux :

Les tableaux permettent d'organiser les widgets à la manière d'un... tableau

N'oubliez pas d'inclure la classe appropriée : **#include <gtkmm/table.h>**

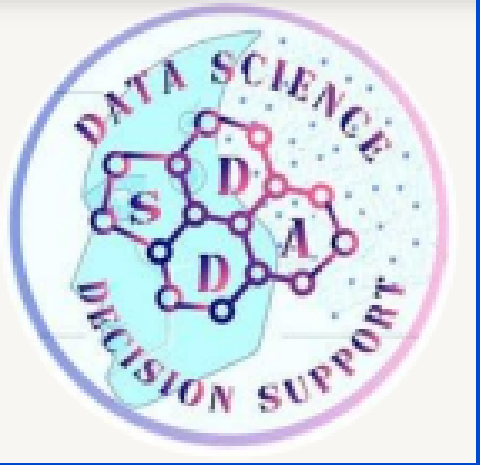
Pour créer un tableau, nous devons lui dire combien de colonnes et de rangées

il aura : **Gtk::Table tableau(1, 2);**



(0, 0) à (1, 1)	(1, 0) à (2, 1)	(2, 0) à (3, 1)
(0, 1) à (1, 2)	(1, 1) à (2, 2)	(2, 1) à (3, 2)
(0, 2) à (1, 3)	(1, 2) à (2, 3)	(2, 2) à (3, 3)





Voyons le prototype de la méthode pour ajouter un bouton :

```
void Gtk::Table::attach(Widget& child, guint left_attach, guint right_attach, guint top_attach, guint bottom_attach, AttachOptions xoptions = FILL|EXPAND, AttachOptions yoptions = FILL|EXPAND, guint xpadding = 0, guint ypadding = 0);
```

child est le widget que nous voulons ajouter au tableau

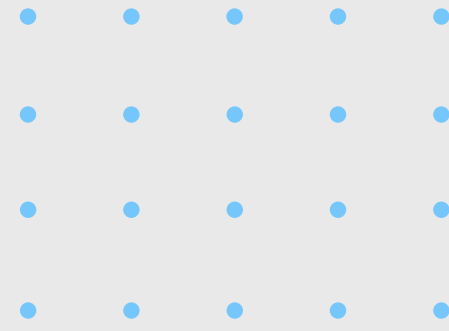
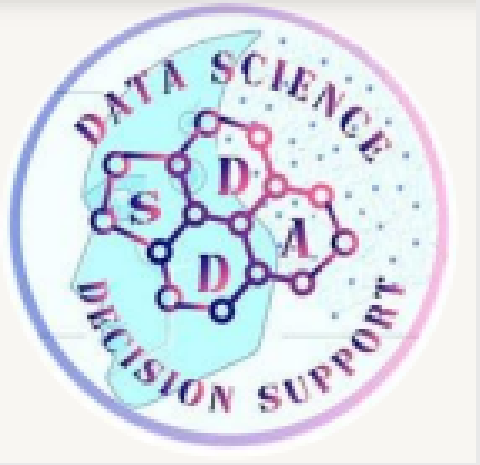
left_attach est la coordonnée x1 (le côté gauche du widget)

top_attach est la coordonnée y1 (le côté haut du widget)

xoptions décrit le comportement du widget lorsqu'il est redimensionné horizontalement

yoptions décrit le comportement du widget lorsqu'il est redimensionné verticalement



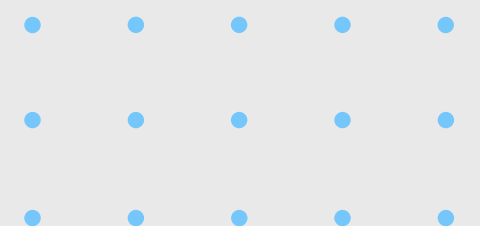


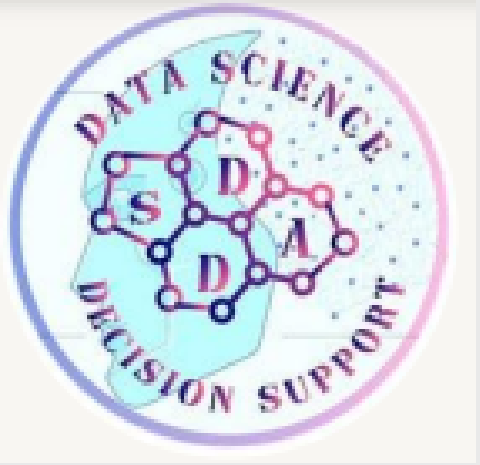
Mais avant, je vais vous dire ce que nous pouvons donner comme xoptions et yoptions!

Gtk::FILL : le widget prend l'espace qui lui est alloué

Gtk::EXPAND : le widget prend l'espace non utilisé autour de lui

Gtk::SHRINK : le widget prend le moins d'espace possible





Exemple :

```
#include <gtkmm/button.h>
#include <gtkmm/main.h>
#include <gtkmm/table.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);

    Gtk::Window fenetre;

    Gtk::Table tableau(1, 2); //Création d'un tableau de 1 case par 2 cases, dont les cases ne sont pas nécessairement de même grandeur.
    tableau.resize(2, 2); //Redimensionner le tableau (2 par 2).
    fenetre.add(tableau);

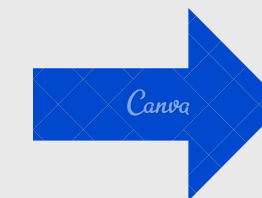
    Gtk::Button bouton1("Bouton 1");
    Gtk::Button bouton2("Bouton 2");
    Gtk::Button bouton3("Bouton 3");
    bouton1.set_can_focus(false);
    bouton2.set_can_focus(false);
    bouton3.set_can_focus(false);

    tableau.attach(bouton1, 0, 1, 0, 1, Gtk::SHRINK, Gtk::SHRINK); //Ajout d'un widget dans le tableau
    tableau.attach(bouton2, 1, 2, 0, 1); //Le widget prend l'espace qui lui est alloué et tout autre espace additionnel.
    tableau.attach(bouton3, 0, 2, 1, 2, Gtk::FILL, Gtk::FILL, 25, 10); //Le widget prend l'espace qui lui est alloué.

    fenetre.show_all(); //L'habituel show_all() pour afficher tous les widgets.

    Gtk::Main::run(fenetre);

    return 0;
}
```





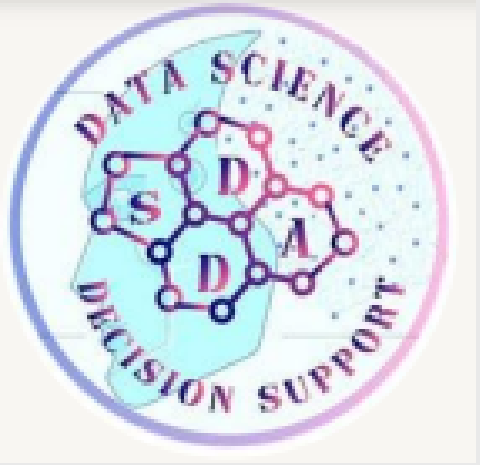
Les Barres de Défilement :

Si un widget prend trop de place pour être affiché au complet, nous pouvons utiliser les barres de défilement pour n'afficher qu'une partie de ce widget, mais avec des barres de défilement pour se déplacer sur ce widget.

Avant toute utilisation, il faut inclure la bonne classe :

```
#include <gtkmm/scrolledwindow.h>
```





Exemple :

```
#include <gtkmm/main.h>
#include <gtkmm/scrolledwindow.h>
#include <gtkmm/textview.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);

    Gtk::Window fenetre;
    fenetre.set_border_width(10);

    //Création d'un conteneur ayant des barres de défilement.
    Gtk::ScrolledWindow barresDeDefilement;
    //Afficher les barres de défilement seulement lorsque c'est nécessaire.
    barresDeDefilement.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::POLICY_AUTOMATIC);

    fenetre.add(barresDeDefilement);

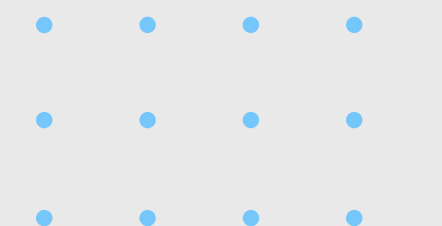
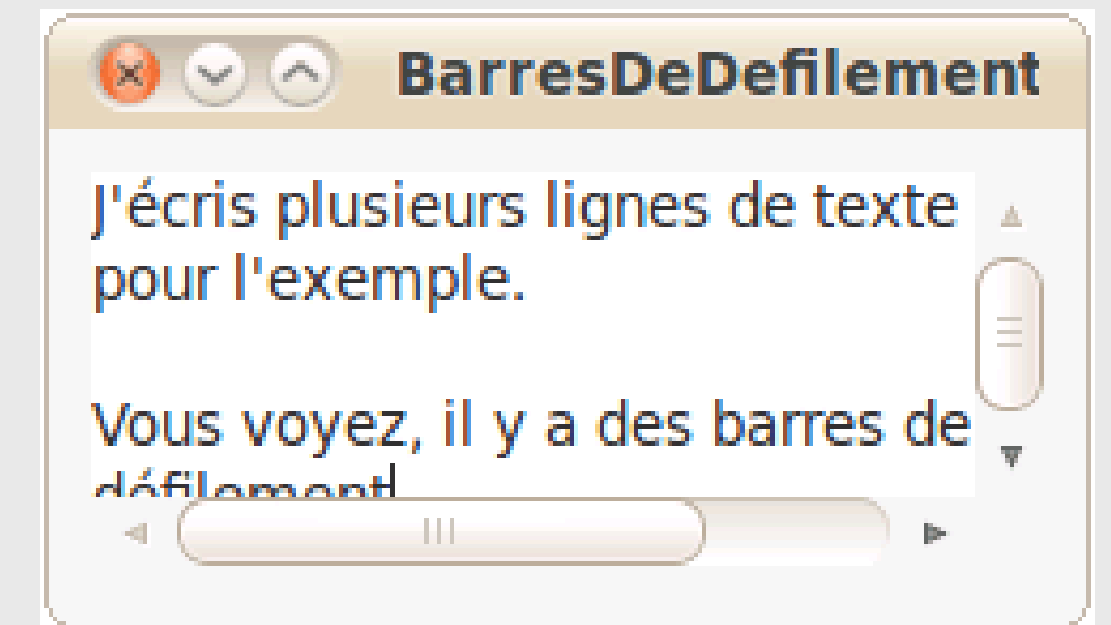
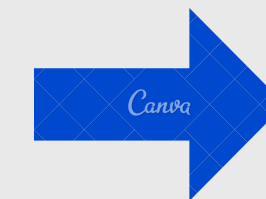
    Gtk::TextView zoneDeTexte; //Création d'une zone de texte.

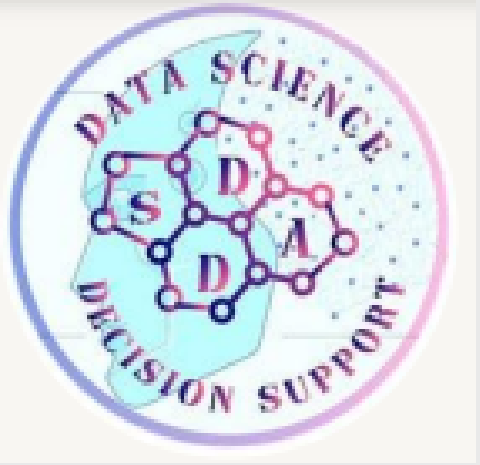
    barresDeDefilement.add(zoneDeTexte); //Ajout de la zone de texte au conteneur.

    fenetre.show_all();

    Gtk::Main::run(fenetre);

    return 0;
}
```





Widgets d'affichage :

L'étiquette :

Une étiquette permet d'afficher du texte, Vous devez inclure l'entête :

```
#include <gtkmm/label.h>
```

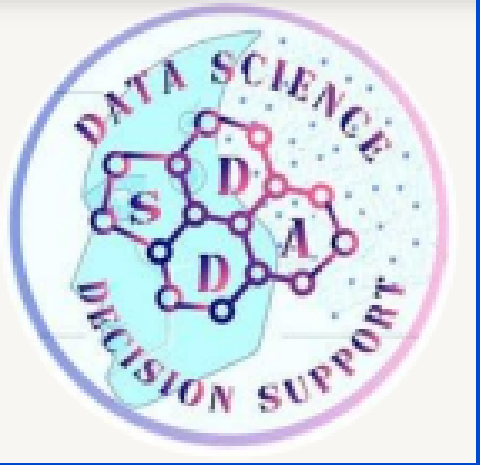
Il est possible de construire une étiquette sans paramètre, comme ceci :

```
Gtk::Label etiquette("Mon texte");
```

Une autre façon de créer une étiquette est de lui envoyer, en plus du texte, les alignements horizontal et vertical du widget

```
Gtk::Label etiquette("Mon texte", Gtk::ALIGN_END, Gtk::ALIGN_START);
```





- **Les Méthodes :**

La méthode **set_selectable()** permet à l'utilisateur de sélectionner le texte :

etiquette.set_selectable();

Voici la fameuse méthode pour centrer le texte :

etiquette.set_justify(Gtk::JUSTIFY_CENTER);

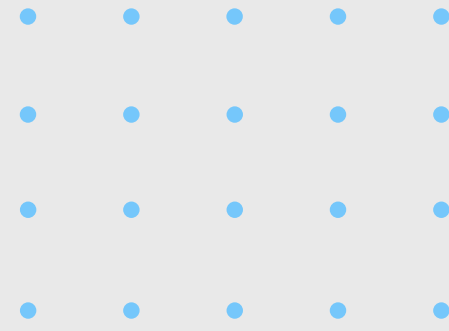
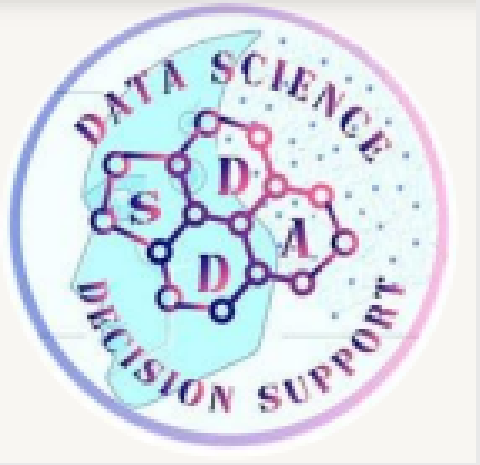
Gtk::JUSTIFY_LEFT : aligner le texte à gauche (par défaut)

Gtk::JUSTIFY_RIGHT : l'aligner à droite

Gtk::JUSTIFY_CENTER : le centrer

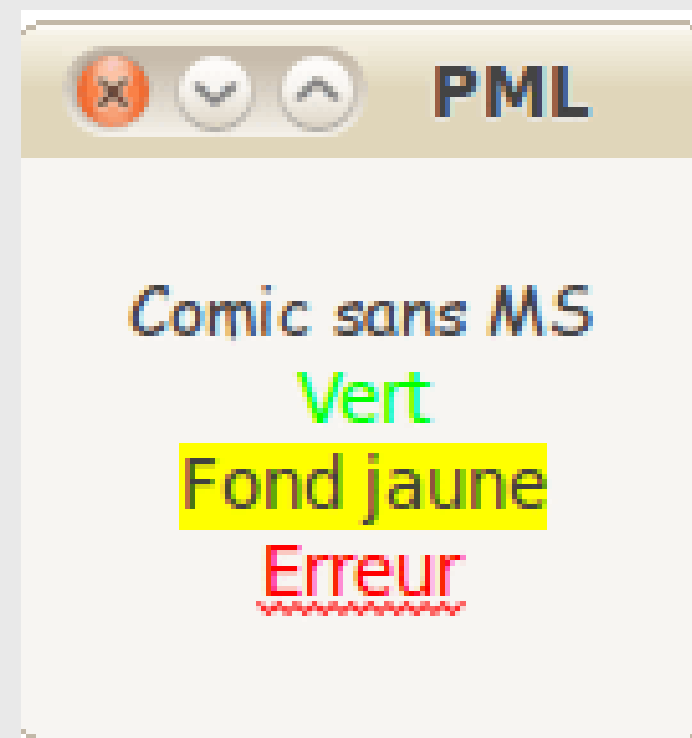
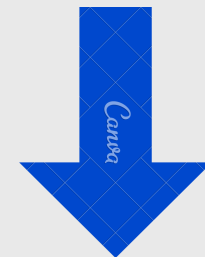
Gtk::JUSTIFY_FILL : le justifier

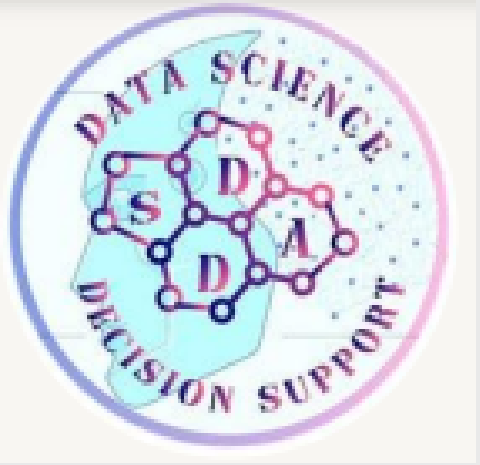




Exemple :

```
etiquette.set_markup("<span face='Comic sans MS'>Comic sans MS</span>\n<span color='green'>Vert</span>\n<span background='yellow'>Fond jaune</span>\n|<span color='red' underline='error'>Erreur</span>");
```





L'image :

Il vous suffit de créer une instance de **Gtk::Image** avec comme seul param le chemin vers le fichier. Ensuite, il ne vous reste qu'à la mettre dans un conteneur et l'afficher.

Exemple :

```
#include <gtkmm/image.h>
#include <gtkmm/main.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);
    Gtk::Window fenetre;

    Gtk::Image image("gtk.png"); //Création d'une image à partir d'un fichier.
    fenetre.add(image);
    image.show();

    Gtk::Main::run(fenetre);
    return 0;
}
```





Il y a un autre moyen de créer une image qui pourrait vous intéresser :
créer une image à partir d'un Stock Item.

Pour ce faire, vous devez donner deux paramètres au constructeur :
le Stock Item et la taille de l'image.

Voici les constantes à utiliser pour la taille de l'image :

Gtk::ICON_SIZE_DND

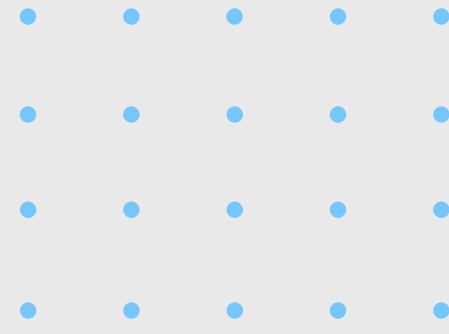
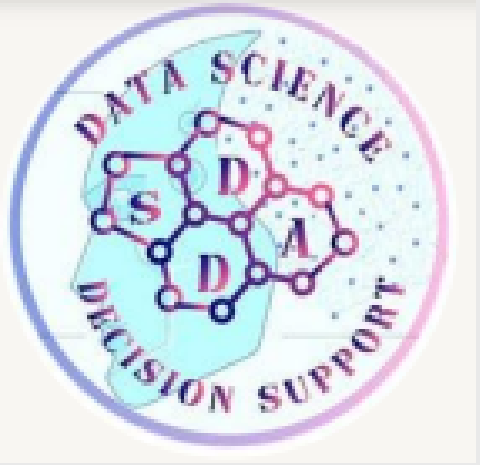
Gtk::ICON_SIZE_MENU

Gtk::ICON_SIZE_SMALL_TOOLBAR

Gtk::ICON_SIZE_LARGE_TOOLBAR

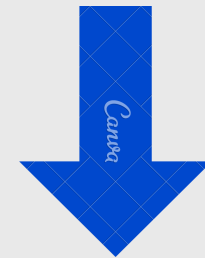
Gtk::ICON_SIZE_BUTTON

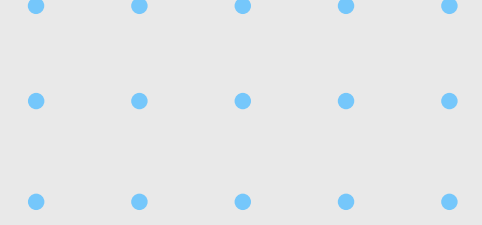
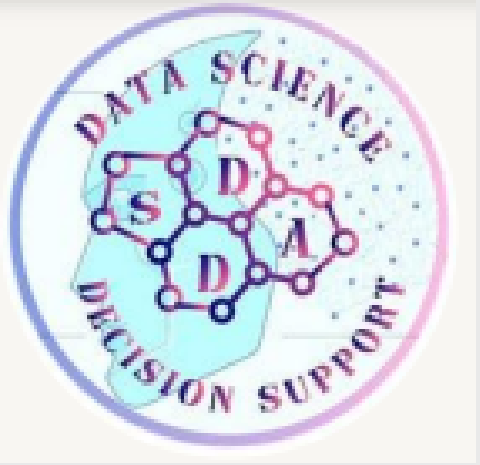




Exemple :

```
Gtk::Image image(Gtk::Stock::QUIT, Gtk::ICON_SIZE_DIALOG);
```





Widgets de saisie :

La zone de texte uniligne :

Cette zone de texte permet à l'utilisateur d'entrer une seule ligne de texte.

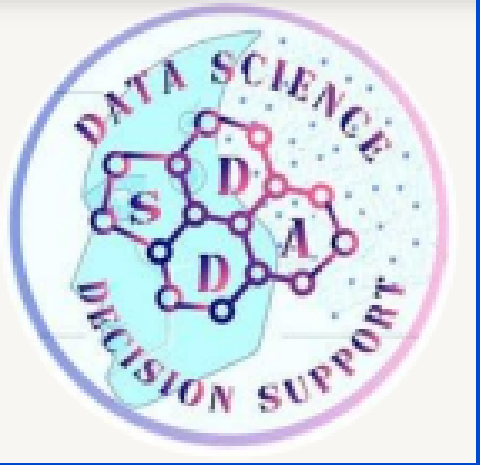
N'oubliez pas d'inclure :

Gtk::Entry zoneTexte

Le constructeur d'une zone de texte est très simple, il ne prend aucun paramètre :

Gtk::Entry zoneTexte





Les Méthodes :

La méthode **set_text()** permet de modifier le texte présent dans la zone de texte:

```
zoneTexte.set_text("Salut les Zér0 !");
```

Une méthode qui peut vous être utile est **set_alignment()**

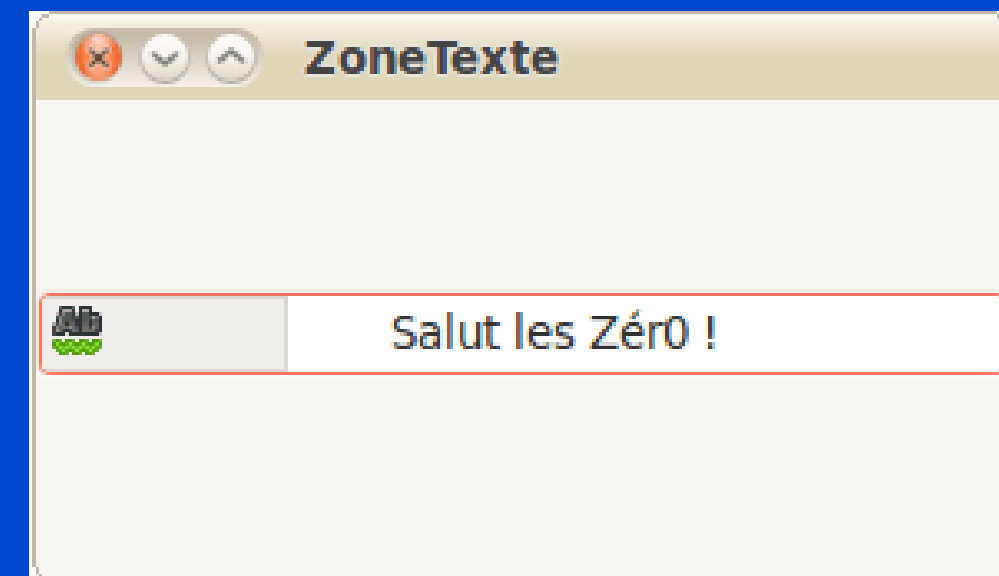
```
zoneTexte.set_alignment(Gtk::ALIGN_CENTER);
```

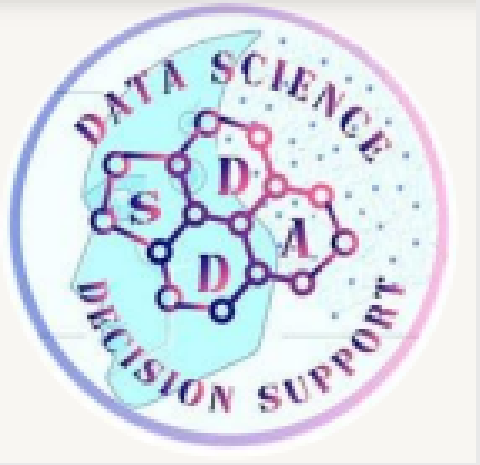
Cette méthode modifie l'alignement horizontal du texte dans la zone de texte.

Gtk::ALIGN_START : à gauche (0

Gtk::ALIGN_CENTER : au centre (0.5)

Gtk::ALIGN_END : à droite (1)





La liste déroulante :

La liste déroulante permet à l'utilisateur de choisir un élément d'une liste.



Vous devez inclure :

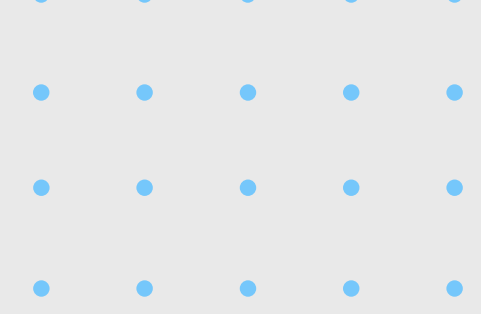
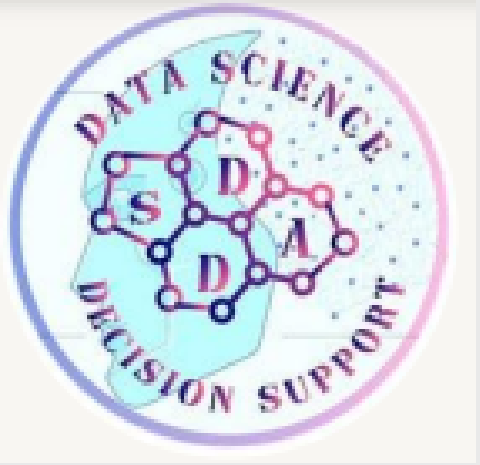
```
#include <gtkmm/comboboxtext.h>
```

Pour construire une liste déroulante, il suffit de créer un objet de type

Gtk::ComboBoxText :

```
Gtk::Entry zoneTexte
```





Les Méthodes :

La première méthode que je vais vous montrer est **append()**.

Celle-ci permet d'ajouter un élément dans la liste déroulante.

```
listeDeroulante.append("France");
```

Une autre méthode utile est **set_active_text()**.

Cette méthode permet de modifier le texte affiché sur la liste déroulante.

Par exemple, si nous voulons que "France" soit l'élément affiché, il faudra faire :

```
listeDeroulante.set_active_text("France");
```





La barre d'état :

La barre d'état permet d'afficher des informations sur l'état de l'application.

Elle est représentée par la classe **Gtk::Statusbar**.

La création d'une telle barre se fait assez facilement : **Gtk::Statusbar barreEtat;**

Nous pouvons ajouter ou enlever des messages dans une barre d'état.

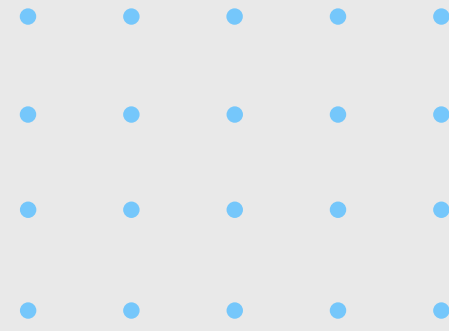
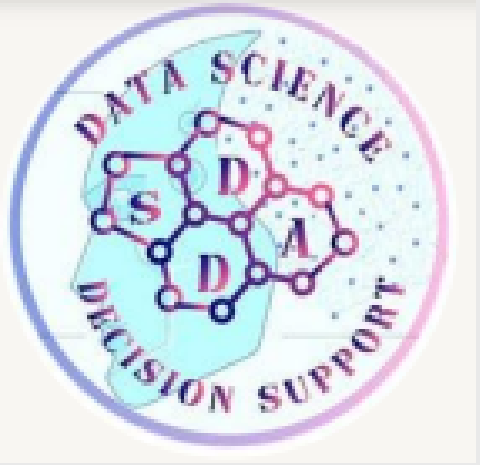
```
int messageID = barreEtat.push("Vous pouvez commencer à utiliser l'application");|
```

Cela est utile s'il vous prend l'envie de supprimer le message :

barreEtat.remove_message(messageID);

Vous pouvez supprimer tous les messages par : **barreEtat.remove_all_messages();**



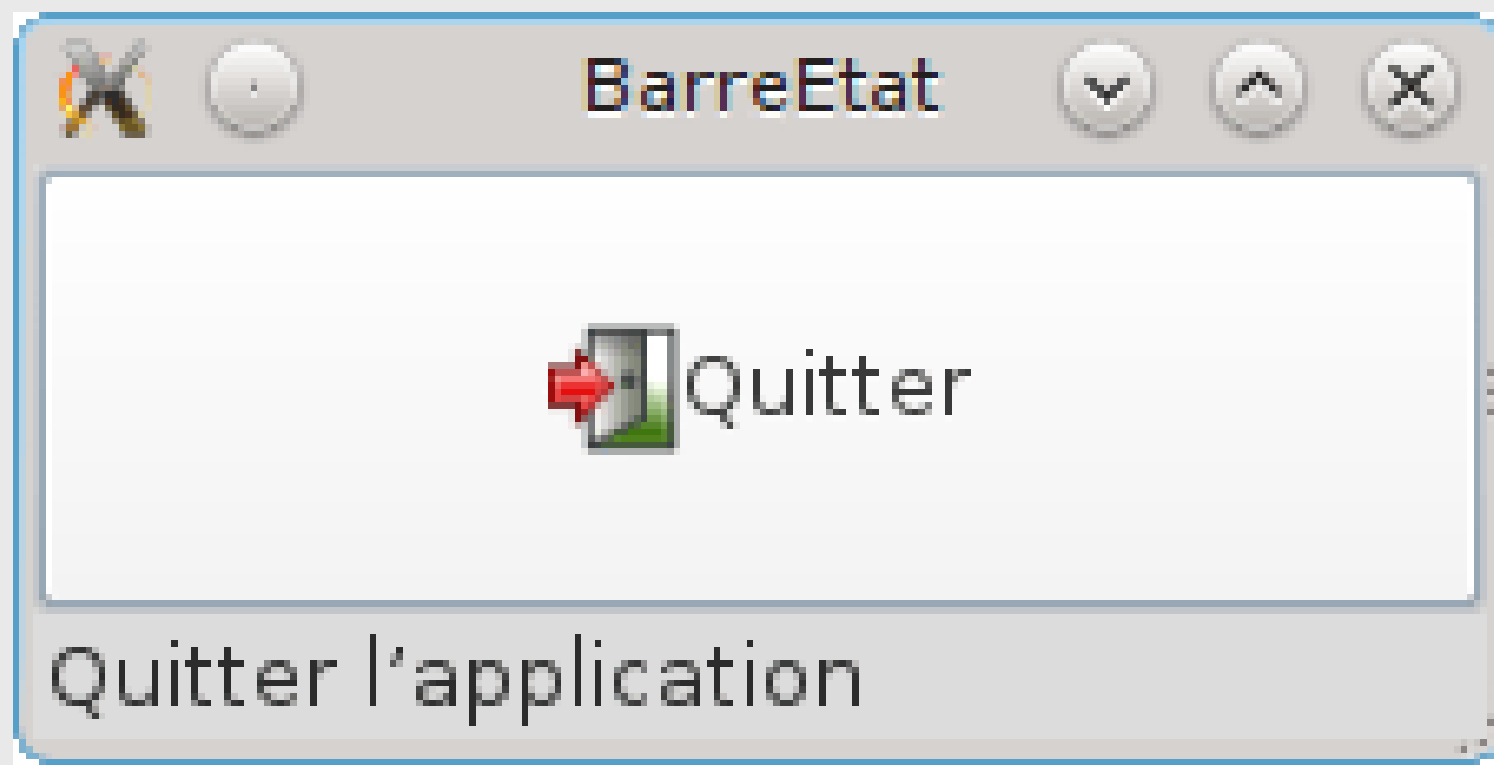


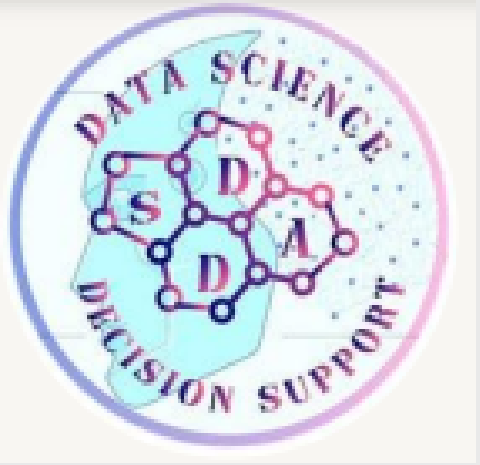
Exemple :

Voici un exemple d'utilisation de la barre d'état.

Ce programme affiche un message lorsque l'utilisateur pointe la souris sur le bouton

Ce message disparaît lorsque le souris ne pointe plus dessus :





Le fichier Fenetre.hpp :

```
#ifndef DEF_FENETRE
#define DEF_FENETRE

#include <string>

#include <gtkmm/box.h>
#include <gtkmm/button.h>
#include <gtkmm/statusbar.h>
#include <gtkmm/stock.h>
#include <gtkmm/window.h>

class Fenetre : public Gtk::Window {
public:
    Fenetre();

    bool afficherMessageEtat(GdkEventCrossing* event, std::string message);
    bool cacherMessageEtat(GdkEventCrossing* event);

private:
    Gtk::Statusbar barreEtat; //Barre d'état.
    Gtk::VBox boiteV;
    Gtk::Button bouton;
};

#endif
```





Le fichier Fenetre.cpp :

```
#include "Fenetre.hpp"

Fenetre::Fenetre() : bouton(Gtk::Stock::QUIT) {
    add(boiteV);

    //Ajouter un message et récupérer son ID.
    int messageID = barreEtat.push("Vous pouvez commencer à utiliser l'application");
    //Enlever le message à partir de son ID.
    barreEtat.remove_message(messageID);

    boiteV.pack_start(bouton);

    //Connexions de signaux au bouton afin d'afficher un message dans la barre d'état lorsqu'il est survolé.
    bouton.signal_enter_notify_event().connect(sigc::bind<std::string>(sigc::mem_fun(*this, &Fenetre::afficherMessageEtat), "Quitter l'application"));
    bouton.signal_leave_notify_event().connect(sigc::mem_fun(*this, &Fenetre::cacherMessageEtat));

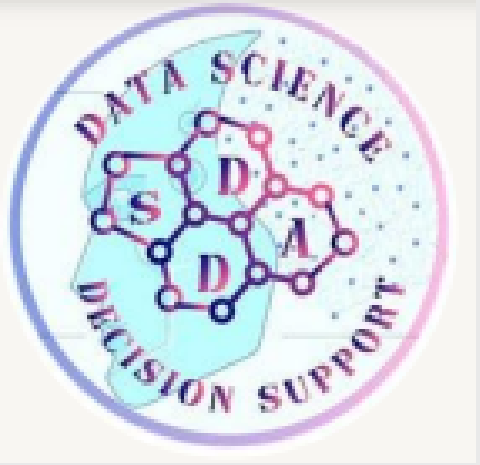
    //Un cas d'utilisation de pack_end() est l'ajout d'une barre d'état.
    boiteV.pack_end(barreEtat, Gtk::PACK_SHRINK);

    show_all();
}

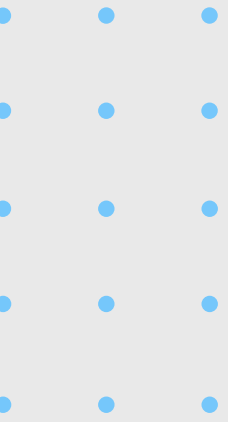
bool Fenetre::afficherMessageEtat(GdkEventCrossing* event, std::string message) {
    (void)event;
    barreEtat.push(message);
    return true;
}

bool Fenetre::cacherMessageEtat(GdkEventCrossing* event) {
    (void)event;
    barreEtat.remove_all_messages();
    return true;
}
```





Les signaux et les fonctions de rappel :



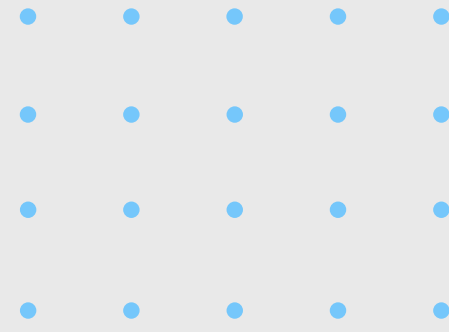
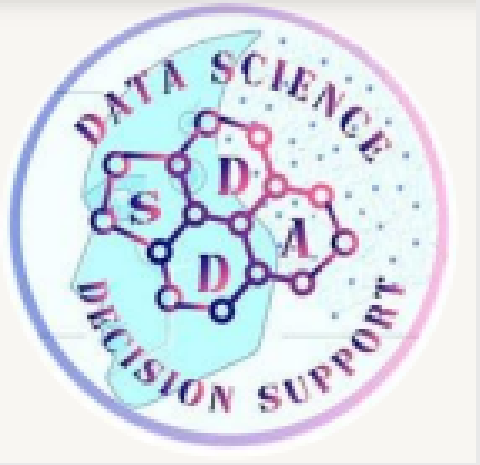
Un signal est un message envoyé par un widget lorsqu'un événement s'est produit sur ce dernier (par exemple, lorsque l'utilisateur clique sur un bouton).

Pour traiter les événements, nous devons connecter un signal à une **fonction de rappel**.

Ce peut être n'importe quelle fonction, qui prend des paramètres ou non.

Ce peut également être une méthode d'une classe.





Connexion d'un signal à une fonction de rappel sans paramètre :

Notre premier exemple sera de connecter le clic sur un bouton à la fermeture du prog

Avant de vous dire comment connecter un signal à une fonction de rappel, je vais vous donner la méthode qui fait arrêter le programme : **Gtk::Main::quit();**

Voici la ligne qu'il faut ajouter : `boutonQuitter.signal_clicked().connect(sigc::ptr_fun(&Gtk::Main::quit));`

Après **boutonQuitter.signal_**, il faut mettre le signal que nous souhaitons connecter à une fonction de rappel.

Ici, c'est **clicked()** (ce qui signifie que ce signal est émis lorsque l'utilisateur clique sur le bouton).



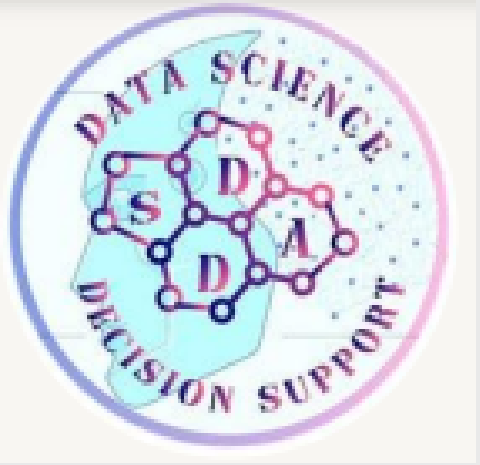


Ensuite, nous connectons (**connect()**) le signal à la fonction de rappel.

Puis, nous indiquons **sigc::ptr_fun()** pour dire que nous connectons le signal à une fonction et non à une méthode.

À la fin, il y a le nom de la fonction, sans parenthèses, précédé d'une esperluette (&).





Voici le code complet :

```
#include <gtkmm/button.h>
#include <gtkmm/main.h>
#include <gtkmm/stock.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);
    Gtk::Window fenetre;

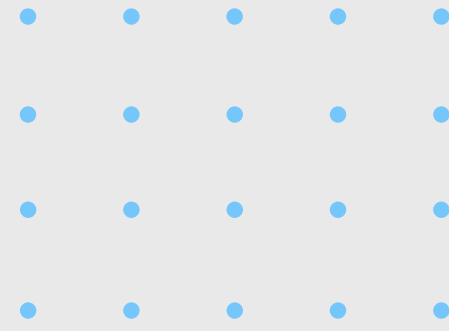
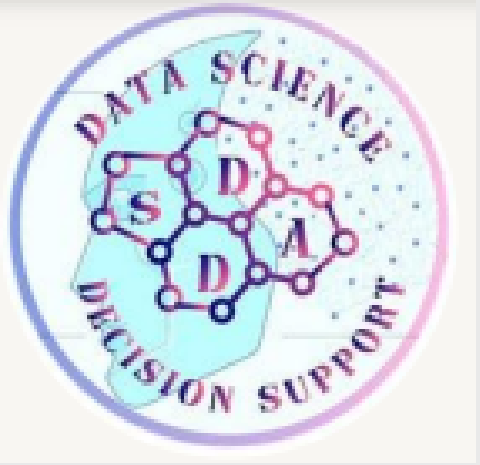
    Gtk::Button bouton(Gtk::Stock::QUIT); //Création d'un bouton.
    fenetre.add(bouton); //Ajout du bouton à la fenêtre.

    bouton.show(); //Ne pas oublier d'afficher le bouton.
    bouton.set_can_focus(false); //Empêcher le bouton d'avoir le focus.

    //Connexion du signal clicked() à la fonction Gtk::Main::quit() qui ferme le programme.
    bouton.signal_clicked().connect(sigc::ptr_fun(&Gtk::Main::quit));

    Gtk::Main::run(fenetre);
    return 0;
}
```





Connexion à une fonction de rappel avec paramètre(s) :

Nous voulons que le clic sur le bouton modifie le titre de la fenêtre.c-à-d appeler **set_title()** avec un argument.Pour cela, il faudra lier le paramètre avec **sigc::bind()**.

Voilà le code : `bouton.signal_clicked().connect(sigc::bind<std::string>(sigc::mem_fun(*this, &Fenetre::set_title), "Titre de la fenêtre"))`

Entre les chevrons se trouve le type du paramètre que nous envoyons.

"**Titre de la fenêtre**" est bien de type **std::string**.

Dans un code complet, ça donne ceci :





```
#include <string>

#include <gtkmm/button.h>
#include <gtkmm/main.h>
#include <gtkmm/window.h>

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);

    Gtk::Window fenetre;
    fenetre.set_title(""); //Cacher le titre de la fenêtre.

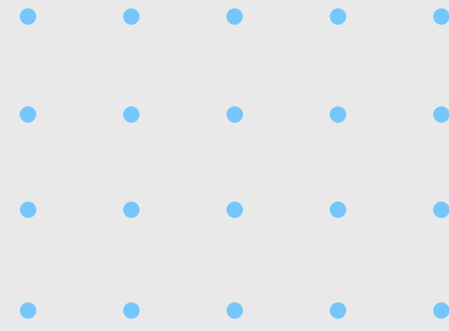
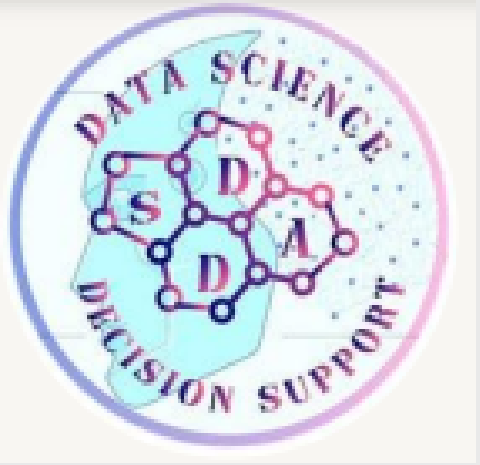
    Gtk::Button bouton("Afficher le titre !");
    fenetre.add(bouton); //Ajout du bouton à la fenêtre.

    bouton.show(); //Ne pas oublier d'afficher le bouton.
    bouton.set_can_focus(false); //Empêcher le bouton d'avoir le focus.

    //Lorsque l'utilisateur cliquera sur le bouton, le titre de la fenêtre changera.
    bouton.signal_clicked().connect(sigc::bind<std::string>(sigc::mem_fun(fenetre, &Gtk::Window::set_title), "Titre de la fenêtre"));

    Gtk::Main::run(fenetre);

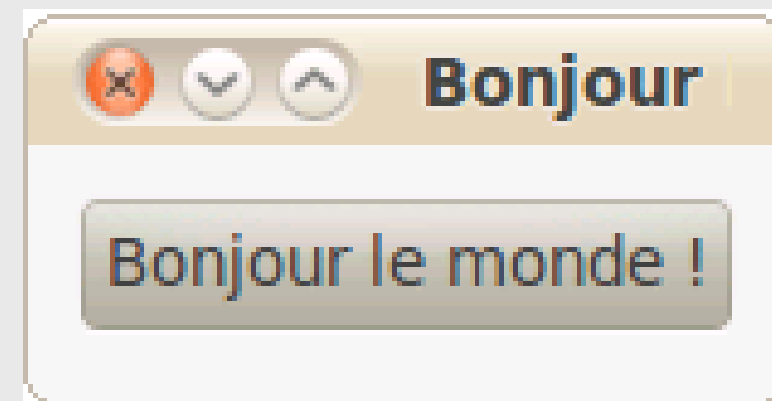
    return 0;
}
```



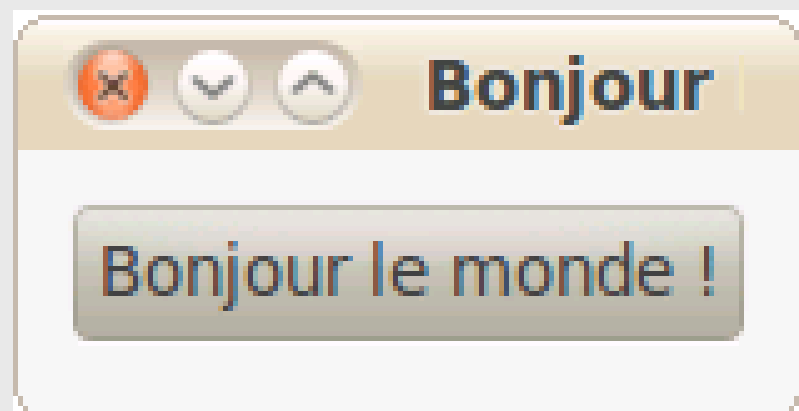
Exemple :

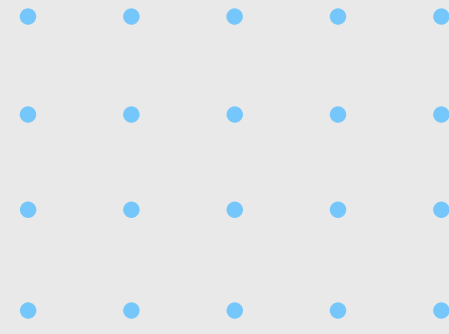
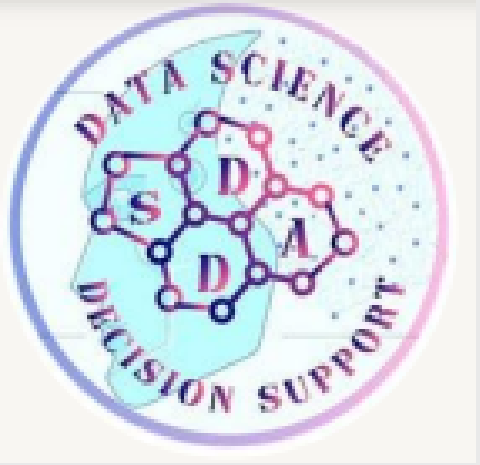
Nous allons concevoir un petit programme qui utilise les signaux.

Le programme affichera un bouton :



Et lorsque l'utilisateur cliquera dessus, nous allons modifier le texte sur le bouton et le titre sur la fenêtre :





Correction:

Le fichier main.cpp

```
#include <gtkmm/main.h>

#include "Fenetre.hpp"

int main(int argc, char* argv[]) {
    Gtk::Main app(argc, argv);
    Fenetre fenetre;
    Gtk::Main::run(fenetre);
    return 0;
}
```



Le fichier Fenetre.hpp

```
#ifndef DEF_FENETRE
#define DEF_FENETRE

#include <string>

#include <gtkmm/button.h>
#include <gtkmm/window.h>

class Fenetre : public Gtk::Window {
    public :
        Fenetre();

    private :
        Gtk::Button bouton;
};

#endif
```





Le fichier Fenetre.cpp

```
#include "Fenetre.hpp"

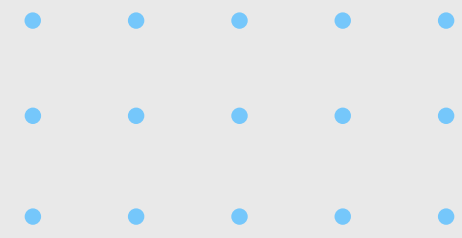
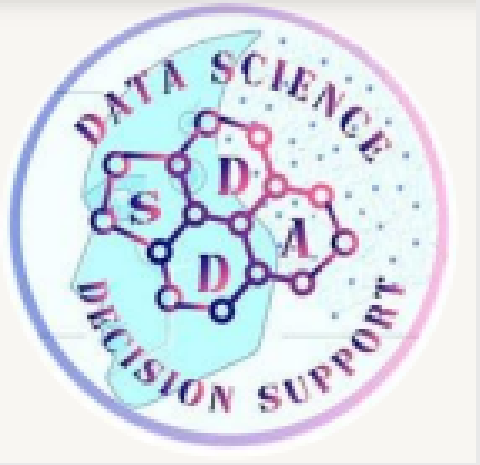
using namespace std;

Fenetre::Fenetre() : bouton("Hello World!") {
    set_title("Hello World!");
    set_border_width(10);

    add(bouton);
    bouton.show();
    bouton.set_can_focus(false);

    //Lorsque l'utilisateur clique sur le bouton, modifier le titre et...
    bouton.signal_clicked().connect(sigc::bind<string>(sigc::mem_fun(*this, &Fenetre::set_title), "Bonjour le monde !"));

    //... le texte sur le bouton.
    bouton.signal_clicked().connect(sigc::bind<string>(sigc::mem_fun(bouton, &Gtk::Button::set_label), "Bonjour le monde !"));
}
```

Créez vos propres fonctions de rappel :

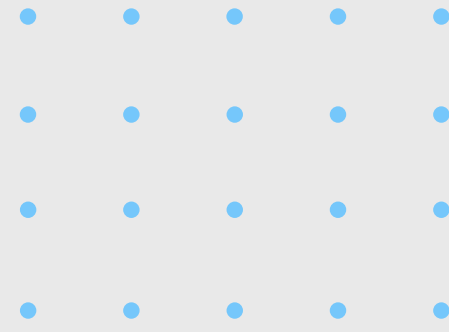
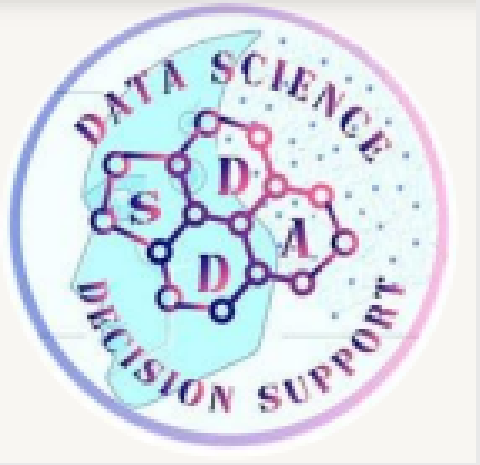
Une fonction de rappel comme les autres. Il renvoie une valeur et a des paramètres.
Pour cette partie, nous allons utiliser une fonction de rappel qui agrandit la taille de la fenêtre. Voici son prototype : **void augmenterTaille();**

Nous allons maintenant écrire son contenu dans le fichier Fenetre.cpp :
Je vous explique ce qui se passe.

```
void Fenetre::augmenterTaille() {  
    int largeur(0);  
    int hauteur(0);  
    get_size(largeur, hauteur);  
    resize(largeur + 10, hauteur + 10);  
}
```

Premièrement, nous créons des variables pour contenir la largeur et la hauteur actuelles de la fenêtre.





Nous obtenons ces données en envoyant ces variables en paramètre à **get_size()**.

Ensuite, nous redimensionnons la fenêtre grâce à la méthode **resize()**

Il ne vous reste plus qu'à connecter un signal à cette fonction et le tour est joué :

```
bouton.signal_clicked().connect(sigc::mem_fun(*this, &Fenetre::augmenterTaille));
```

À chaque clic sur un bouton, la fenêtre s'agrandit de 10px.





La fonction de rappel avec des paramètres :

Modifions le prototype de notre fonction de rappel pour celui-ci :

```
void augmenterTaille(int augmentationLargeur, int augmentationHauteur);
```

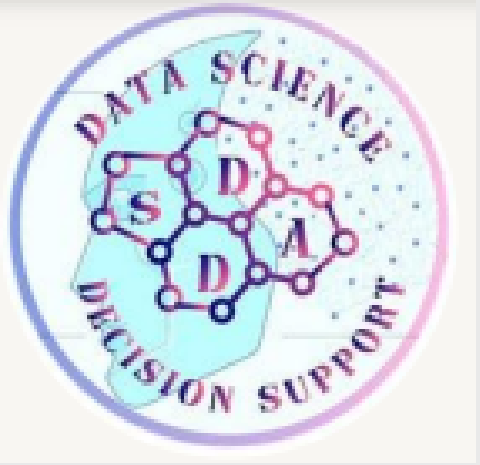
Cette fois, nous allons envoyer à cette fonction la largeur et la hauteur que nous voulons ajouter à la fenêtre.

```
void Fenetre::augmenterTaille(int augmentationLargeur, int augmentationHauteur) {  
    int largeur(0);  
    int hauteur(0);  
    get_size(largeur, hauteur);  
    resize(largeur + augmentationLargeur, hauteur + augmentationHauteur);  
}
```

Et voici comment se connecter à le signal :

```
bouton.signal_clicked().connect(sigc::bind<int, int>(sigc::mem_fun(*this, &Fenetre::augmenterTaille), 130, 140));
```





le code complet :

Le fichier fenetre.hpp

```
#ifndef DEF_FENETRE
#define DEF_FENETRE

#include <gtkmm/button.h>
#include <gtkmm/window.h>

class Fenetre : public Gtk::Window {
public:
    Fenetre();

private:
    Gtk::Button bouton;

    void augmenterTaille(int augmentationLargeur, int augmentationHauteur);
};

#endif
```

Le fichier Fenetre.cpp

```
#include "Fenetre.hpp"

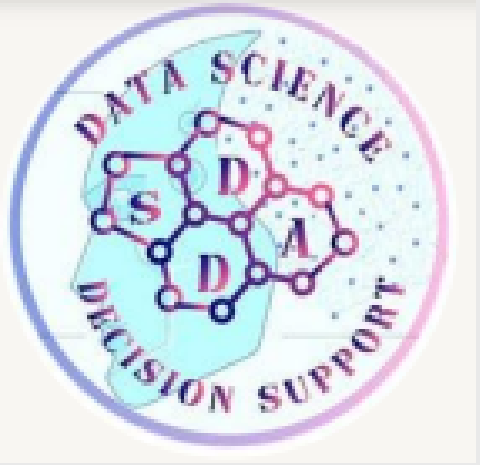
Fenetre::Fenetre() : bouton("Agrandir") {
    set_default_size(200, 200); //Déterminer la taille par défaut de la fenêtre.
    set_border_width(10); //Ajouter une bordure de 10px autour du bouton.
    add(bouton);

    //Connexion du signal clicked() du bouton à la méthode augmenterTaille() de la présente fenêtre.
    //Nous lui passons deux paramètres.
    bouton.signal_clicked().connect(sigc::bind<int, int>(sigc::mem_fun(*this, &Fenetre::augmenterTaille), 130, 140));
    show_all();
}

void Fenetre::augmenterTaille(int augmentationLargeur, int augmentationHauteur) {
    int largeur(0);
    int hauteur(0);
    get_size(largeur, hauteur);
    resize(largeur + augmentationLargeur, hauteur + augmentationHauteur);
}
```

Le fichier main.cpp reste le même, Il n'est pas nécessaire de le réécrire

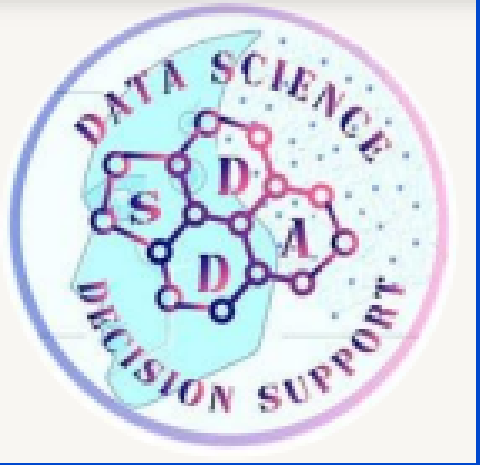




Nous sommes arrivés à la fin du cours, nous espérons que vous avez acquis suffisamment d'informations qui peuvent vous aider pendant votre pratique

N'hésitez pas à laisser votre commentaire





- # Les Références:

<http://sdz.tdct.org/sdz/creer-des-interfaces-graphiques-en-c-avec-gtkmm.html#Crezvospropresfonctionsderappel>

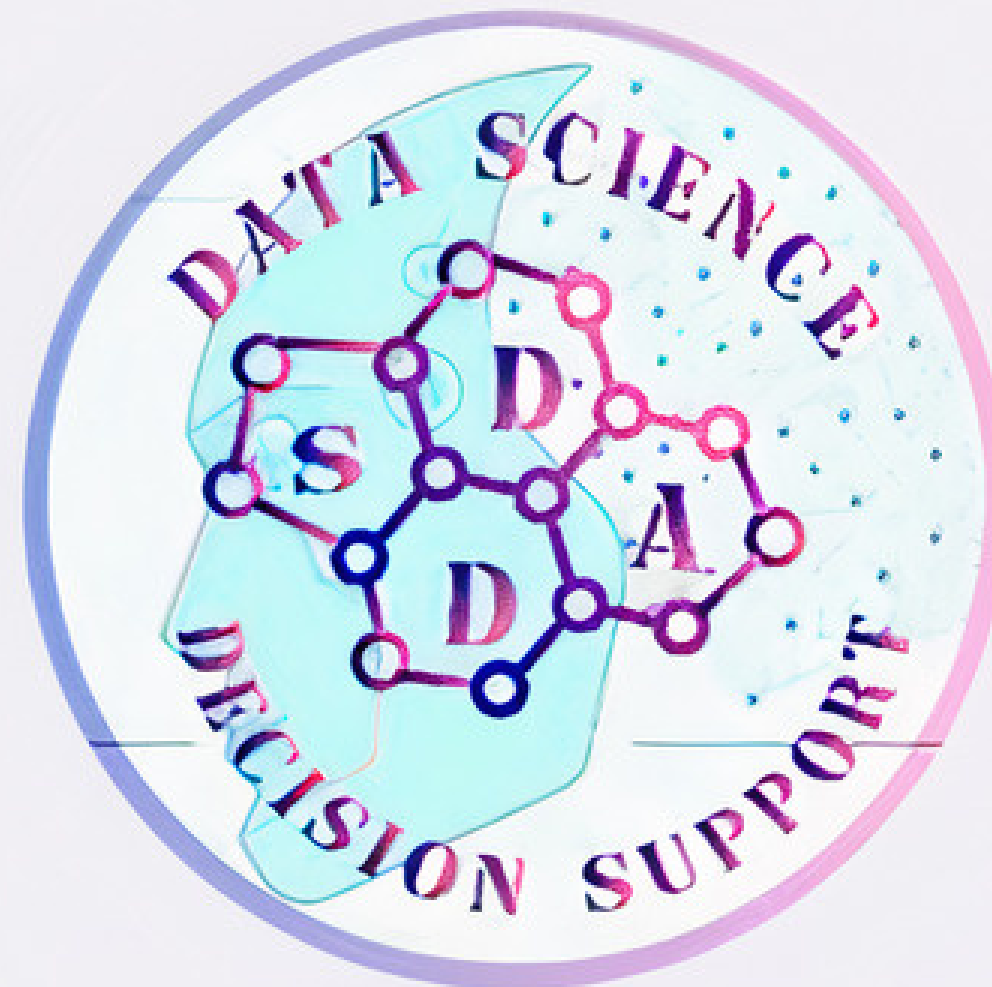
<https://www.lucidarme.me/gtkmm/>

<https://docs.huihoo.com/gtkmm/programming-with-gtkmm-3/3.4.1/en/sec-gtkmm.html>

http://people.irisa.fr/Thomas.Guyet/enseignements/OCI/Tuto_GTKMM.pdf



THANK YOU



www.sdadclub.tech