1 Objectifs

Le but de ce TD/TP est de se familiariser avec les boîtes de dialogue et de message.

2 Éléments de cours

2.1 Présentation

2.1.1 Filiation

GtkWindow → GtkDialog → GtkMessageDialog

2.1.2 Constitution

Une **boîte de dialogue** est une fenêtre préformatée constituée :

- d'une zone de travail (GtkVBox)
  (on y place usuellement des labels et des zones de saisie) ;
- d'une zone de boutons (GtkHBox)
  (les boutons peuvent être insérés au moment même de la création de la boîte ou plus tard,
  par utilisation d'une fonction spécifique) ;
- d'une ligne de séparation (objet GtkHSeparator qui sépare la zone de travail et la zone des boutons).

Une **boîte de message** est une boîte de dialogue préformatée qui contient dans sa zone de travail un message (d'avertissement, d'information etc.). Ce message est défini à la création même de la boîte de message.

2.1.3 Vocation dans une IHM (Interface Homme-Machine)

Classiquement, une application est constituée d'une fenêtre principale avec un menu qui permet d'ouvrir d'autres fenêtres, chaque fenêtre correspondant à une fonctionnalité précise de l'application. Ces fenêtres peuvent être des GtkDialog qui permettent à l'utilisateur d'atteindre ses buts (ex. : s'identifier auprès d'un système, paramétrer l'application, interroger une base de données etc.). De plus, lorsque l'utilisateur interagit avec l'application, celle-ci doit le tenir informé du bon usage qu'il doit faire du logiciel ainsi que de l'état de ses actions (ex. : message d'information en cas de saisie incomplète, message d'erreur en cas d'erreur logicielle, barre d'avancement d'une tâche en cours etc.) : c'est le rôle des boîtes de message.

2.1.4 Principales fonctions

`GtkWidget* gtk_dialog_new(void);` création
`GtkWidget* gtk_dialog_add_button(GtkDialog* pDialogue, const gchar* libelle, gint valRetour);` ajoute un bouton de libellé "libelle" à la boîte de dialogue "pDialogue" le paramètre "valRetour" définit la valeur retournée par le bouton lorsque l'on clique sur celui-ci le pointeur renvoyé n'est autre que l'adresse du bouton qui vient d'être créé (utile si l'on veut agir que le bouton après sa création)
`void gtk_dialog_run(GtkDialog* pDialogue);` lance la boîte de dialogue pDialogue (équivalent de la fonction "gtk_main()" pour une fenêtre principale)
`void gtk_widget_destroy(GtkDialog* pDialogue);` détruit la boîte de dialogue (... attention : toujours penser à détruire la boîte de dialogue pour libérer la mémoire !)

2.1.5 Manipulation

`GtkWidget* pDialogue;` déclaration comme GtkWidget*... *as usual!*
`GtkWidget* pVBox = GTK_DIALOG(pDialogue)→vbox;` récupération de la GtkVBox de la boîte de dialogue dans la variable pVBox la GtkVBox représente la zone de travail de la boîte de dialogue
`gtk_widget_show_all(GTK_DIALOG(pDialogue)→vbox);` *ou* `gtk_widget_show_all(pDialogue);` permet d'afficher la zone de travail et tout ce qu'elle contient lors du lancement de la boîte par `gtk_Dialog_run()`

2.1.6 En résumé

Pour travailler sur la zone de travail de la fenêtre, il faut :
- accéder à la GtkVBox de la fenêtre "GTK_DIALOG(pDialogue)→vbox ;"
- utiliser ensuite les méthodes habituelles des objets GtkVBox "gtk_box_pack_start();",...

Pour travailler sur la zone de boutons de la fenêtre, il faut :
- créer les boutons par les méthodes définies au niveau des objets GtkDialog "gtk_dialog_new_with_buttons();", "gtk_dialog_add_button();".

Le cycle de vie d'une fenêtre :
- lancement par "gtk_dialog_run() ;"
- affichage par appel préalable à "gtk_widget_show_all() ;", sur la fenêtre elle-même ou sa zone de travail ;
- fin d'affichage après action de l'utilisateur sur la fenêtre : clic sur les boutons de la zone de boutons, fermeture de la fenêtre ;
- destruction (libération de l'espace mémoire) par appel à "gtk_widget_destroy();".

2.1.7 Ajout d'un bouton

Pour ajouter un bouton avec la méthode "`gtk_dialog_add_button();`", il faut spécifier le libellé (ou libellé avec raccourci clavier, ou image) du bouton (deuxième argument de la fonction), ainsi que la valeur retournée par le bouton (troisième argument de la fonction). Cette valeur peut être une valeur définie par l'utilisateur, ou une constante du type énuméré GtkResponseType. Pour revenir sur le libellé du bouton, on peut écrire directement la chaîne de caractéres souhaitée (ex. : "Quitter", "_Quitter"), ou l'identité d'une image en stock (ex. : GTK_STOCK_QUIT).

Exemple d'utilisation :

```
gtk_dialog_add_button(GTK_DIALOG(pDialogue), GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL);
```

2.1.8 Gestion événementielle de la boîte de dialogue

Les événements survenus à la boîte de dialogue sont gérés au niveau de la fenêtre appelante (ou plus précisément, de la fonction callback qui, suite à une action sur un objet de la fenêtre appelante, l'aura créée). Lorsque l'on active la boîte de dialogue avec la fonction "`gtk_dialog_run()`", celle-ci s'affiche, puis se fermera par action de l'utilisateur sur un bouton ou sur la fenêtre (comportement par défaut : le programmeur peut aussi forcer la fermeture de la fenêtre dans le code). Si la fenêtre est fermée suite à un "clic" sur un bouton, c'est la valeur retournée par ce bouton qui sera renvoyée par la fonction "`gtk_dialog_run()`" (GTK_RESPONSE_CANCEL dans notre exemple). Si la fenêtre est fermée suite à un "clic" sur le coin N-E de celle-ci, c'est la valeur retournée par la fenêtre qui sera renvoyée (par défaut, la fenêtre renvoie alors la valeur GTK_RESPONSE_DELETE_EVENT).

2.1.9 Petit exemple avant de passer aux choses sérieuses

Cet exemple ouvre une boîte de dialogue, dont le titre est "Veux-tu dialoguer avec moi ?", et qui comporte un label "Cliquer sur le bouton de votre choix", ainsi que les deux boutons "Oui" et "Non".

```
// Fonction callback attachée à un signal survenu à un objet quelconque
void boiteDialogue(GtkWidget* pWidget, gpointer pData)
{
    // Déclaration des objets
    GtkWidget* pDialogue = NULL;
    GtkWidget* pLabel = NULL;
    GtkWidget* pVBox;
    guint reponse = GTK_RESPONSE_NONE;
    // Création des objets
    pLabel = gtk_label_new("Cliquez sur le bouton de votre choix");
    pDialogue = gtk_dialog_new();
    // Apparence des objets
    // (En qualité d'héritier de GtkWindow, les GtkDialog peuvent utiliser les fonctions définies sur les GtkWindow)
    gtk_window_set_title(GTK_WINDOW(pDialogue), "Veux-tu dialoguer avec moi ?");
    gtk_window_set_default_size(GTK_WINDOW(pDialogue), 300, 100);
    // Spécification du contenu de la GtkDialog
    pVBox = (GTK_DIALOG(pDialogue))→vbox;
    gtk_box_pack_start(GTK_BOX(pVBox), pLabel, FALSE, FALSE, 0);
    gtk_dialog_add_button(GTK_DIALOG(pDialogue), GTK_STOCK_YES, GTK_RESPONSE_YES);
    gtk_dialog_add_button(GTK_DIALOG(pDialogue), GTK_STOCK_NO, GTK_RESPONSE_NO);
    // Affichage des objets
    gtk_widget_show_all(pVBox);
    // Lancement de la GtkDialog et gestion de sa valeur retour
    reponse = GTK_RESPONSE_YES;
    while (reponse == GTK_RESPONSE_YES)
    {
        reponse = gtk_dialog_run(GTK_DIALOG(pDialogue));
    }
    // Destruction de la GtkDialog
    gtk_widget_destroy(pDialogue);
}
```

Dans cet exemple, on lance autant de fois la boîte de dialogue que l'utilisateur clique sur le bouton "Oui". En revanche, dès qu'il ferme la fenêtre (gtk_dialog_run() renvoie alors la valeur GTK_RESPONSE_DELETE_EVENT) ou qu'il clique sur le bouton "Non" (gtk_dialog_run() renvoie alors la valeur GTK_RESPONSE_NO), la variable "reponse" contient alors une valeur différente de "GTK_RESPONSE_YES" et la boîte n'est pas relancée (condition d'arrêt de la boucle "while").

2.1.10 Boîtes de message

`GtkWidget* gtk_message_dialog_new(GtkWindow* parent, GtkDialogFlags flags, GtkMessageType type, GtkButtonsType buttons, const gchar* message);`
`GtkWindow* parent` : fenêtre à partir de laquelle la GtkMessageDialog est lancée. Comme on lancera des fenêtres avec l'option `GTK_DIALOG_MODAL`, la paramètre `parent` peut être mis à NULL.
`GtkDialogFlags flags` : décrit le comportement de la GtkMessageDialog. Utiliser `GTK_DIALOG_MODAL` : on ne peut rien faire tant que la GtkMessageDialog est ouverte
`GtkMessageType type` : nature du message (alerte, erreur, information etc.). Le texte et l'icône correspondants seront affichés dans la barre de titre de la GtkMessageDialog. Valeurs possibles : GTK_MESSAGE_{INFO, WARNING, QUESTION, ERROR}.
`GtkButtonsType buttons` : bouton ou combinaison de boutons à afficher dans la GtkMessageDialog. Valeurs possibles : GTK_BUTTONS_{NONE, OK, CLOSE, CANCEL, YES_NO, OK_CANCEL}. Ex. : avec GTK_BUTTONS_NONE, la GtkMessageDialog ne comporte pas de bouton ; avec GTK_BUTTONS_OK_CANCEL, la GtkMessageDialog comporte deux boutons, "OK" et "Annuler".
`message` : contenu du message à l'intention de l'utilisateur. Le message peut contenir des variables (même syntaxe que printf), ex. : "`gtk_message_dialog_new(..., "i = %d", i);`".

Exemple : pour forcer l'utilisateur à répondre "Oui" sur la GtkDialog de l'exemple précédent, on peut ouvrir une boîte de message avertissant l'utilisateur qu'il doit répondre "Oui", tant qu'il répond "NON".

```
// Fonction de création de GtkMessageDialog à laquelle on passe en argument la fenêtre appelante
guint boiteMessage(GtkWidget* parent)
{
    // Déclaration des objets
    GtkWidget* pMessage = NULL;
    guint reponse = GTK_RESPONSE_NONE;
    // Création des objets
    pMessage = gtk_message_dialog_new(GTK_WINDOW(pData), GTK_DIALOG_MODAL,
            GTK_MESSAGE_WARNING, GTK_BUTTONS_OK, "Vous devez dire Oui !");
    // Lancement de la GtkMessageDialog
    reponse = gtk_dialog_run(GTK_DIALOG(pMessage));
    // Destruction de la GtkMessageDialog
    gtk_widget_destroy(pMessage);
    return reponse;
}

// Dans la fonction de callback "boiteDialogue"
guint reponseMessage = GTK_RESPONSE_NONE;
guint reponseDialogue = GTK_RESPONSE_YES;
while ((reponseDialogue == GTK_RESPONSE_YES) || (reponseDialogue == GTK_RESPONSE_NO))
```

```
{
    reponseDialogue = gtk_dialog_run(GTK_DIALOG(pDialogue));
    if (reponseDialogue == GTK_RESPONSE_NO)
    {
        reponseMessage = boiteMessage(pDialogue);
    }
}
```

Dans cet exemple, on ne traite pas la valeur retour de la boîte de message (bien qu'on la récupère, pour la forme), puisque celle-ci n'a aucune incidence sur la suite des opérations. En effet, quelle que soit la façon dont la boîte de message s'arrête ici, cet arrêt a pour seule conséquence que de redonner la main à la fenêtre appelante, c'est-à-dire à la boîte de dialogue.

3 Exercices

Le but de l'exercice est d'implémenter une fonctionnalité d'une application ; cette fonctionnalité consiste tout simplement à saisir le login et mot de passe de l'utilisateur, de sorte à ce que celui-ci passe en mode connecté (ex. : passage du mode extranet au mode intranet sur un site internet). On ne va pas développer l'application, aussi celle-ci consistera en le minimum dont on a besoin pour lancer une boîte de dialogue à partir d'un bouton, c'est-à-dire : en une fenêtre (la fenêtre principale) et un bouton.

1. Créer une fenêtre, de taille 600×200, qui contient 1 bouton. Le titre de la fenêtre est "Application lambda - utilisateur non connecte", le label du bouton "S'identifier".
2. Faire en sorte que, lorsque l'on clique sur le bouton "S'identifier", une boîte de dialogue s'affiche. Celle-ci contient 2 zones de saisie (l'une pour le login, l'autre pour le mot de passe) et deux boutons, le premier, "Ok", qui renvoie GTK_RESPONSE_OK et le second, "Annuler", qui renvoie GTK_RESPONSE_CANCEL. Le titre de cette boîte est "S'identifier".
3. Attention : le mot de passe ne doit pas apparaître à l'écran lorsqu'il est saisi ! De plus, la boîte de dialogue doit être d'une taille de 300×100. Enfin, les deux zones de saisie sont limitées à 8 caractères.
4. Faire en sorte, au signal "activate" de chaque zone de saisie, de vérifier la longueur des textes saisis : si cette longueur n'est pas dans l'intervalle [6, 8] (au moins 6 caractères, au plus 8 caractères), avertir l'utilisateur par l'intermédiaire d'une boîte de message. Cette boîte de message, de type avertissement, indique précisément à l'utilisateur qu'il faut que sa saisie contienne entre 6 et 8 caractères. Le comportement de cette boîte est GTK_DIALOG_MODAL ; enfin, cette boîte contient le seul bouton "Ok".
5. On suppose que l'on dispose d'une fonction "gboolean verifierLogin(const gchar* login, const gchar* passwd);", qui vérifie (dans une base de données par exemple) que le login et le mot de passe passés en paramètres correspondent bien à un utilisateur connu. Vérifier alors, lorsque l'utilisateur aura appuyé sur le bouton "Ok" de la boîte de dialogue "S'identifier", que la saisie de l'utilisateur est correcte (correspond bien à un utilisateur connu). Si c'est le cas, l'utilisateur est renvoyé sur la fenêtre principale, dont le titre devient : "Application lamda : utilisateur connecte". Sinon, l'utilisateur doit reprendre la saisie... aprés avoir fermé une nouvelle boîte de message qui lui indique son erreur et met à vide les zones de saisie de la boîte de message.