

Programming GTK in Free Pascal: Menus

Florian Klämpfl
and
Michaël Van Canneyt

September 2000

1 Introduction

In the third article on programming the GTK toolkit, the use of menus in GTK is explored.

Menus can be built in essentially 2 ways; the easier way through the use of a itemfactory, and the more complex way, doing all necessary calls manually. The advantages of both ways are discussed.

2 Menus the easy way: The item factory

The easy way to construct a menu is to use an item factory. An Item factory gets as input an array of records, which describe a menu structure, and returns a completely built menu, ready to be added to a window.

The great advantage of an item factory is that it is easy to use; a disadvantage is that

1. There is less control over the produced menu items; e.g. displaying a menu item with a small icon is not possible.
2. The callbacks of the constructed menu is different from the usual signal model, making it difficult to combine a menu entry with a speedbutton. There are also 2 types of callback, so type checking is not possible.
3. In Pascal, constant records must be specified using the names of the members; this makes the array with the menu items to be rendered quite complicated.

To create a menu, first the item factory must be created. The function to do this is defined as follows:

```
gtk_item_factory_new ( container_type : TGtkType ;  
                      path : Pgchar ;  
                      accel_group : PGtkAccelGroup ) : PGtkItemFactory ;
```

The three arguments to this function have the following meaning:

container_type This identifies the kind of menu that will be rendered. It can have one of the following values:

GTK_MENU_BAR_TYPE A menu bar will be created to hold all items.

GTK_MENU_TYPE A menu that can be used as a popup menu, or that can be attached as a sub-menu to another menu, will be created.

GTK_OPTION_MENU_TYPE Makes everything in a drop-down style menu which can be used to select one value.

path is the name of the menu to be generated.

accel_group Is a pointer to a group of accelerators. All accelerators for the generated menu will be attached to this group.

The accelerator group needed for the item factory can be constructed using a simple call to `gtk_accel_group_new`; this function takes no arguments, and returns a pointer to a new accelerator group.

To actually create the menu, a call to `gtk_item_factory_create_items` is needed; This procedure is defined as follows:

```
gtk_item_factory_create_items ( ifactory : PGtkItemFactory ;
                               n_entries : guint ;
                               entries : PGtkItemFactoryEntry ;
                               callback_data : gpointer );
```

The first argument to this call, `ifactory`, is the itemfactory; the second argument, `n_entries`, is the number of items in the array of records describing the menu. The third argument, `entries`, is the actual array describing the menu. The last argument `callback_data` is a pointer that will be passed to the menu callbacks.

The menu structure that should be created by the item factory is an array of records of the type `TGtkItemFactoryEntry`. This record is defined as follows:

```
TGtkItemFactoryEntry = record
  path : Pgchar;
  accelerator : Pgchar;
  callback : TGtkItemFactoryCallback;
  callback_action : guint;
  item_type : Pgchar;
end;
```

The fields have the following meaning:

path The first entry is the path of the menu item. This indicates the place of the menu entry in the whole menu. For instance, the menu item **New** in the menu **File** would be designated by `'/ File/_New'` the slash is used to separate the menu levels.

To make one of the letters of the menu item name active, so the item can be selected by pressing the letter (on the keyboard) when the menu is opened, the key to be used should be preceded by an underscore. In e.g. `'/ File/_New'` the letter **N** could be used to select the **New** item if the **File** menu is active.

accelerator To make a shortcut to the menu item so it can be activated at all times, the shortcut name can be specified in the `accelerator` field. This can be any key, together with some modifiers. e.g. `'<control>_N'` make the key combination 'CTRL-N' a shortcut.

The accelerator should be specified as normal text. A list of possible modifiers can be found in table 1.

Table 1: List of modifier strings for shortcut keys

Modifier	alias
<control>	<ctl>, <ctrl>
<shift>	<shft>
<alt>	<mod1>

callback Contains a pointer to the function that should be called when the menu item is activated. The type of the menu handler is not the same as a normal signal handler; The actual callback should be of the type `TGtkItemFactoryCallback1`:

```
procedure ( callback_data : gpointer ;
            callback_action : guint ;
            widget : PGtkWidget ); cdecl ;
```

Which is not the same as the type of the `callback` field, so a typecast will always be necessary.

callback_action This is passed on to the callback in the `callback_action` parameter.

item_type is the type of menu item. Several types can be used; the complete list can be found in 2, but the most important ones are `'<Item>'` which specifies a normal menu item, and `'<Branch>'` which indicates a sub-menu.

Table 2: Possible menu item types

Item type	Menu kind
'<Item>'	indicates a normal item. An empty string or Nil have the same meaning.
'<CheckItem>'	a check menu item.
'<ToggleItem>'	a toggle menu item (same as check menu).
'<RadioItem>'	a radio item.
'<Separator>'	a separator bar.
'<Branch>'	an item to hold a submenu.
'<LastBranch>'	an item to hold a submenu, but right aligned.

Now all elements to create a menu are introduced, and the menu can be created. The following definitions should now be clear:

Var

```
Window   : PGtkWidget ;
MenuBar  : PGtkWidget ;
```

Type

```
FC = TGtkItemFactoryCallback ;
```

Const

```
NrMenuItems = 21 ;
TheMenu : Array [1..NrMenuItems] of TGtkItemFactoryEntry = (
  (path : '/_File' ; Accelerator : Nil ;
   Callback : Nil ; Callback_action : 1 ; item_type : '<Branch>'),
  (path : '/ File/_New' ; Accelerator : '<ctrl>N' ;
   Callback : FC(@Menu) ; Callback_action : 1 ; item_type : Nil ),
  { ... }
```

Here the `FC` type is introduced to make the typecast of the `Menu` handler easier; the `TheMenu` constant is not given completely, since it is too long and not instructive. The complete structure can be found in the sources accompanying this article.

Using the above definitions, the menu can now be constructed:

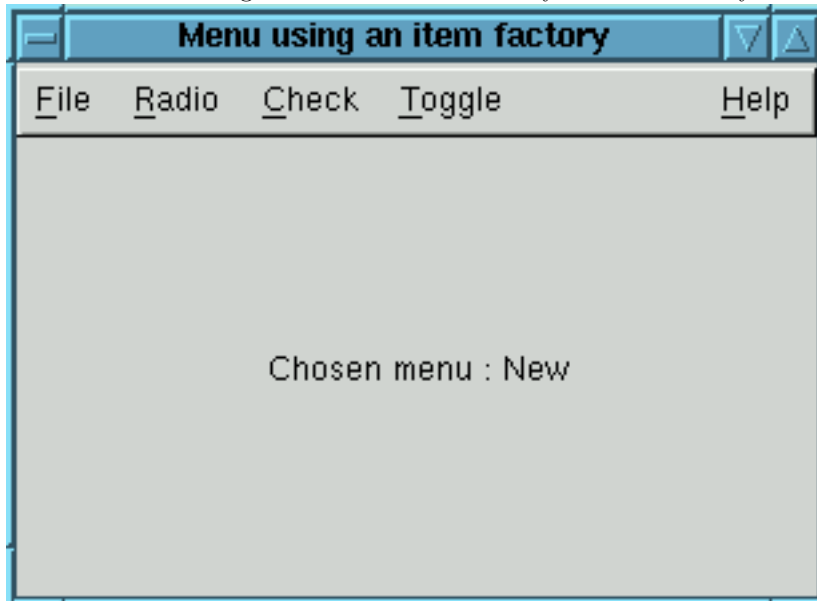
Procedure MakeMenu ;

Var

```
Factory : PGtkItemFactory ;
Accel    : PGtkAccelGroup ;
```

begin

Figure 1: The menu made by the item factory.



```

Accel:=gtk_accel_group_new ;
Factory := gtk_item_factory_new (GTK_MENU_BAR_TYPE, '<main>', accel );
gtk_item_factory_create_items (Factory, NrMenuItems, @TheMenu, Nil );
gtk_window_add_accel_group (GTK_Window(Window), accel );
MenuBar:=gtk_item_factory_get_widget ( Factory, '<main>');
end;

```

The `gtk_window_add_accel_group` call attaches the accelerator group that was filled up by the item factory to the window.

The `gtk_item_factory_get_widget` call finally fetches the object created by the item factory and stores it in a widget variable.

The Menu callback used in the menus is defined as follows:

```

procedure menu(Data : GPointer;
                Action : Guint;
                Widget : pGtkWidget); cdecl;

```

```

Var
  TheLabel : PgtkWidget;
  LabelText : Pchar;
  S : AnsiString;

```

```

begin
  TheLabel:=g_list_nth_data (
    gtk_container_children (
      GTK_CONTAINER(Widget)),0);
  gtk_label_get (gtk_Label (theLabel), @LabelText);
  S := 'Chosen menu : ' + Strpas (LabelText);
  gtk_label_set_text (GTK_LABEL (DisplayLabel), pchar(S));
end;

```

The `DisplayLabel` is a label located on the window, it is used to give some feedback on the used menu. The code to extract the menu name from the menu widget passed to the handler will be explained later on.

The result of all this is shown in figure 1.

As can be seen from the code above, the creation of a menu using an item factory

in GTK is not so hard. The drawback of the above method lies mainly in the fact that Pascal handles constant records differently than C, which makes the array that describes the menu structure rather difficult to read.

The second drawback is that there is little control over the created items.

3 Menus the hard way: manually

When creating menus manually, mainly 4 objects are involved:

- The menu items themselves. To a menu item, a menu can be associated, creating a sub-menu.
- Menus, which contain a collection of menu items,
- A accelerator group. This will be used to keep a collection of shortcut keys for the menu items.
- A menu bar, which can hold several menu items and their associated menus.

The last object is optional, if e.g. a pop-up menu is wanted.

To create a menu in a window, the following steps are involved:

1. Create an accelerator group. The accelerator group should be connected to the window.
2. Create a menu bar, and attach it to the window.
3. For each menu that should appear in a menu bar, do the following:
 - Create a menu item, which will be shown in the menu bar.
 - Create a menu to hold the items that should pop up when the menu is activated.
4. To each menu created in the previous step, add as many menu items as needed. Add an accelerator to the group created in step 1.

To make these steps easier (each of them involves quite some calls to GTK functions) some functions will be introduced that make this easier.

The first function is the most simple one; it attaches a separator to a menu:

Function AddSeparatorToMenu (Menu : PgtkMenu) : PgtkMenuItem ;

begin

```
Result := pgtkmenuItem ( gtk_menu_item_new );
gtk_menu_append ( Menu, pgtkWidget ( result ) );
gtk_widget_show ( PgtkWidget ( result ) );
```

end;

The function takes one parameter, Menu, the menu to which the separator will be attached. A separator is created by simply creating an empty menu item. Creating a new (empty) menu item is done with the `gtk_menu_item_new` call.

With the `gtk_menu_append` call, the newly created item is then added to the menu. Lastly, the item is shown; it will not become actually visible till the menu is activated. If this is omitted, it will also not be visible when the menu is activated.

Adding a menu with a shortcut key to a menu is a little more involved. Some more elements are needed:

1. The menu to which to attach the menu item.
2. The accelerator group to which the accelerator key should be added.

3. The caption of the menu. An underscore character will indicate the letter of the menu that will be used as a shortcut to activate the item.
4. The shortcut for the menu item. An empty string means no shortcut.
5. A callback function which will be called when the menu item is activated, and callback data which will be sent to the callback.

All these elements are found in the declaration of the following function:

```
Function AddItemToMenu ( Menu : PGtkMenu;
                        ShortCuts : PGtkAccelGroup;
                        Caption : AnsiString;
                        ShortCut : AnsiString;
                        CallBack : TgtkSignalFunc;
                        CallBackdata : Pointer
                        ) : PGtkMenuItem;
```

Var

```
Key, Modifiers : guint;
LocalAccelGroup : PGtkAccelGroup;
TheLabel : PGtkLabel;
```

begin

The variables declared in this function will be explained as the code is presented.

First of all, a menu item must be created. Since a caption for the menu is provided, the `gtk_menu_item_new_with_label` will be used to create a menu that has a label as a child:

```
Result:=pgtkmenuItem(gtk_menu_item_new_with_label(''));
TheLabel:=GTK_LABEL(GTK_BIN(Result)^.child);
Key:=gtk_label_parse_uline(TheLabel,Pchar(Caption));
```

After the menu item is created, the child label is fetched. The label caption is then set using the `gtk_label_parse_uline` function. This function will search a text for underscore characters, remove them from the text, and then set the label's caption with the result. All letters which had an underscore character prepended will be underlined in the label.

The function returns the first letter that had an underscore prepended. It is stored, so it can be used to make an accelerator:

```
If Key<>0 then
  begin
    LocalAccelGroup:=gtk_menu_ensure_uline_accel_group(Menu);
    gtk_widget_add_accelerator(PGtkWidget(result),'activate_item',
                              LocalAccelGroup,Key,
                              0,TGtkAccelFlags(0));
  end;
```

The call to `gtk_menu_ensure_uline_accel_group` returns the accelerator group associated with the menu. If no group existed yet, one will be created. The `gtk_widget_add_accelerator` call takes the following parameters:

- A pointer to a widget to which the accelerator should be attached.
- The name of the signal which will be triggered when the shortcut is activated.
- The accelerator group to which the shortcut should be installed, usually this will be the accelerator group for the window to which the widget is attached, but in this case this is the accelerator group of the menu (which will only be active when the menu is actually shown)
- The key from the shortcut.

- The modifiers that should be pressed together with the key. For the menu, this should be 0, since just the key should be hit.
- The accelerator flags.

After the menu item was created and its underlined key was made into an accelerator, the menu can be attached to the menu:

```
gtk_menu_append(Menu, pgtkWidget(result));
```

If a shortcut key was passed along to the procedure, can be added to the window's accelerator group with the following code:

```
If (ShortCut<>'') and (ShortCuts<>Nil) then
  begin
    gtk_accelerator_parse (pchar(ShortCut), @key, @modifiers);
    gtk_widget_add_accelerator(PGtkWidget(result), 'activate-item',
                              ShortCuts,Key,
                              modifiers, GTK_ACCEL_VISIBLE);
  end;
```

The call to `gtk_accelerator_parse` will parse a string which describes a shortcut key, and returns the corresponding key and modifiers, which can then be passed on to the `gtk_widget_add_accelerator` call.

After the accelerator has been installed, the only thing that remains to be done is to connect the callback to the activation of the menu:

```
If Callback<>Nil then
  gtk_signal_connect(PGtkObject(result), 'activate',
                    Callback, Callbackdata);
gtk_widget_show(PgtkWidget(result));
end;
```

As the last line in the procedure, the newly created menu item is shown. If the menu isn't visible yet, this will do nothing, but will ensure that the item is also visible when the menu is visible.

Now a menu-item and a separator can be added to a menu. What remains to be done is to add a menu to a menu bar. This is done in the following procedure, which is given in its entirety:

```
Function AddMenuToMenuBar(MenuBar : PGtkMenuBar;
                          ShortCuts : PGtkAccelGroup;
                          Caption : AnsiString;
                          Callback : TgtkSignalFunc;
                          Callbackdata : Pointer;
                          AlignRight : Boolean;
                          Var MenuItem : PgtkMenuItem
                          ) : PGtkMenu;
```

```
Var
```

```
  Key : guint;
  TheLabel : PGtkLabel;
```

```
begin
```

```
  MenuItem:=pgtkmenuItem(gtk_menu_item_new_with_label(''));
  If AlignRight Then
    gtk_menu_item_right_justify(MenuItem);
  TheLabel:=GTK_LABEL(GTK_BIN(MenuItem)^.child);
  Key:=gtk_label_parse_uline(TheLabel, Pchar(Caption));
  If Key<>0 then
    gtk_widget_add_accelerator(PGtkWidget(MenuItem), 'activate-item',
                              Shortcuts,Key,
                              GDK_MOD1_MASK, GTK_ACCEL_LOCKED);
  Result:=PGtkMenu(gtk_menu_new);
```

```

If CallBack<>Nil then
    gtk_signal_connect(PGtkObject(result), 'activate',
                      Callback, Callbackdata);
    gtk_widget_show(PgtkWidget(Menuitem));
    gtk_menu_item_set_submenu(Menuitem, PgtkWidget(Result));
    gtk_menu_bar_append(MenuBar, PgtkWidget(Menuitem));

```

The code for this procedure quite similar as the previous one. The main differences are:

- The result is not a menuitem, but a whole menu. The menuitem that is displayed in the menu bar itself is returned in the `Menuitem` parameter.
- The shortcut key for the underlined item is added to the window's accelerator group, and has the ALT key (or Mod1) as the modifier key.
- the created menu is attached to the menu item as a sub menu, and it is the menu-item which is attached to the menu bar.

With the above calls, a menu can be constructed with a simple set of calls:

```

FileMenu:=AddMenuToMenuBar(MenuBar, accel, '_File', Nil,
                          Nil, False, TempMenuitem);
AddItemToMenu(FileMenu, accel, '_New', '<control>N',
              TgtkSignalFunc(@menu), DisplayLabel);
AddItemToMenu(FileMenu, accel, '_Open', '<control>O',
              TgtkSignalFunc(@menu), DisplayLabel);
AddItemToMenu(FileMenu, accel, '_Save', '<control>S',
              TgtkSignalFunc(@menu), DisplayLabel);
AddSeparatorToMenu(PGtkMenu(FileMenu));
AddItemToMenu(FileMenu, accel, '_Quit', '<control>Q',
              TgtkSignalFunc(@destroy), Nil);
{ ... }

```

The complete list of calls to create the menu can be found in the sources accompanying this article.

The second program is of course bigger than the first, due to all the code to create the menus. Nevertheless, the manual way of creating has its advantages: it's quite easy to extend the `AddItemToMenu` to add a bitmap to the menu entry as well. Using a itemfactory, there is (currently) no way to add images to a menu.

Adding a bitmap to a menu is quite easy, and requires only a few extra lines of code. The key point is that the `gtkmenuitem` object is just an empty container (it descends from `gtkbin`), which does not display anything by itself. The `gtk_menu_item_new_with_label` call creates a menu item and puts a `gtklabel` in it to display the menu item caption. Instead of a label object, almost any other object can be put in the item. This fact is used in the following code to add a bitmap in front of the menu caption, in a new procedure to be called `AddImageItemToMenu`:

```

Result:=pgtkmenuitem(gtk_menu_item_new);
hbox:=PGtkHBox(gtk_hbox_new(false,0));
gtk_container_add(pgtkcontainer(result),pgtkWidget(hbox));
pixmap:=gdk_pixmap_create_from_xpm(Nil,@BitmapData,Nil,pchar(Bitmap));
Image := PgtkPixMap(gtk_pixmap_new(Pixmap, BitmapData));
gtk_box_pack_start(PGtkBox(hbox),pgtkWidget(image),false,false,0);
TheLabel:=PgtkLabel(gtk_label_new(''));
gtk_box_pack_start(PGtkBox(hbox),pgtkWidget(TheLabel),True,True,0);
Key:=gtk_label_parse_uline(TheLabel,Pchar(Caption));

```

In the first line, a plain menu item is created with `gtk_menu_item_new`. In the following two lines, a `GTKHBox` is added to the menu item, and a reference to the box is stored in the `hbox` variable.

Then, a pixmap is created from a filename. The filename is passed in the `BitMap` parameter to our routine. Using the newly created pixmap, an `Image` is created, which can then be added to the box.

Finally, a regular GTK label is created to hold the caption of the menu item, and added to the box. After that the procedure continues as for a normal menu.

The complete code for the above `AddImageItemToMenu` routine can be found in the sources of the third example, accompanying this article. The result can be seen in figure 2

Figure 2: The menu with bitmaps



Some notes regarding this algorithm are in order:

1. It would be possible to have not a filename passed to the routine, but directly pass a pixmap object as well; for instance when using a toolbar, toolbuttons corresponding to the menu entries could share the same pixmaps as the menu entries.
2. Some alignment issues may arise when the menu contains items with and without bitmaps. The above code does not address these issues. To solve them, the regular menu items should also be constructed e.g. using a `hbox` or a table with an empty cell. Also, an algorithm to determine whether any item of the menu has an image would be needed.
3. The shortcut key is no longer shown in the menu widget; The reason for this is unknown to the authors of this article; unfortunately the lack of documentation on GTK prevents the implementation of a remedy.
4. The menu callback can no longer retrieve the menu text using a straightforward approach, since the label displaying the caption is no longer the only child widget of the menu item. The callback has been adapted for this in the example.

Taking into account the above arguments should make it possible to write better menu-creating routines which would replace the item factory completely, and which would enable the use of bitmaps in menu items.