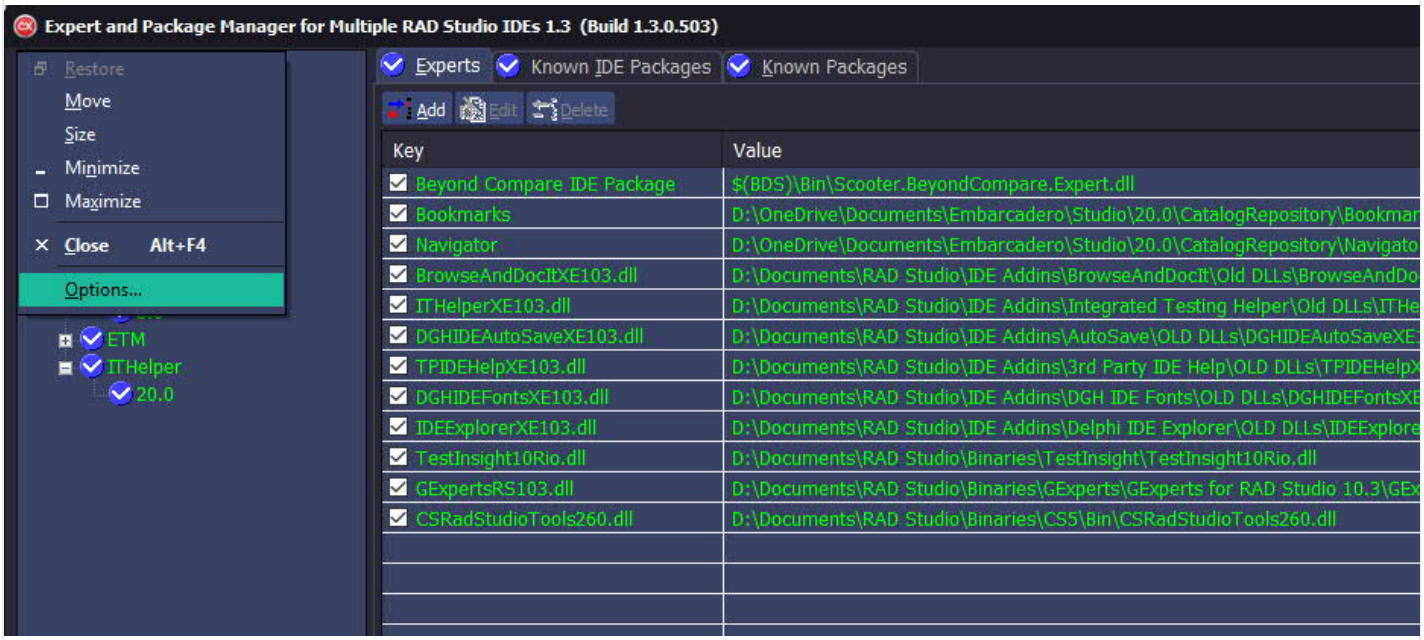
Dave's Development Blog

Software Development using Borland / Codegear / Embarcadero RAD Studio

Adding a menu item to the System Menu

By David | June 20, 2019 0 Comment

I have a few application which have minimal user interfaces and I needed to add an options dialogue to allow the user the change the theme for the application and also some of the custom colours for the information being shown. Being a minimalistic interface I did not want to add a menu bar, toolbar or place a button in a strange place on the interface so I thought I'd have a go at adding a menu to the system menu.



I have to say that I did do a quick Google search to get me started and found the following article which help however I was working C++ Builder not Delphi so things were not so straightward (but I've provided the Delphi code below as well).

https://stackoverflow.com/questions/12659300/adding-a-shortcut-to-a-programmatically-added-system-menu-option

Create the menu

The first thing to do is create a method in which we create the menu item. Why a method? Well you'll see why later on when we find a problem 😕

First we need to create a const integer value that we will use to uniquely identify our menu item.

In C++ I defined the following

```
constexpr int iMenuID = 170;
```

In Delphi I define it as a class constant as follows:

```
Type
  TfrmSystemMenu = class(TForm)
  ...
  private
    const
      iMenuID = 170;
  ...
  end;
```

In C++ Build I wrote the following method to create the menu item.

```
void TfrmExpertManager::AddOptionsMenu() {
   HMENU SystemMenu = GetSystemMenu(Handle, False);
   AppendMenu(SystemMenu, MF_SEPARATOR, 0, NULL);
   AppendMenu(SystemMenu, MF_STRING, iMenuID, L"&Options…");
 };
```

In Delphi we write the following method to create the menu item.

```
procedure TfrmSystemMenu.AddOptionsMenu;
var
  SysMenu : HMENU;
begin
  SysMenu := GetSystemMenu(Handle, False);
  AppendMenu(SysMenu, MF_SEPARATOR, 0, '');
  AppendMenu(SysMenu, MF_STRING, iMenuID, '&Options…');
end;
```

Both versions of the function do the same thing. First they get the handle of the system menu associated with the given form handle and then first add a separator to the end of the menu and then the new menu item with our ID above and a caption.

Handle the menu click

The next task is to handle the menu click. Unlike the VCL, these menus are not assigned an event handler, we need to intercept the `WM_SYSCOMMAND` windows message and check one of its parameters for our custom menu ID. In Delphi this is relatively easy (and I'll show you) but I was using C++ Builder and message handing is a little more tricky. In the form header file we need to define our message handler and bind it using a macro to the windows message as follows:

```
class TfrmExpertManager : public TForm
```

```
 {
 __published: // IDE-managed Components
    ...
 private: // Constants
 private:
    ...
    void __fastcall WMSysCommand(TWMSysCommand &amp;Message);
    BEGIN_MESSAGE_MAP
      MESSAGE_HANDLER(WM_SYSCOMMAND, TWMSysCommand, WMSysCommand)
    END_MESSAGE_MAP(TForm)
    ...
 public:      // User declarations
    ...
};
```

The method declaration is straightforward but note that there are no message handling extra keyword here as you would do in Delphi. The macro below (and it must be afterwards) binds the `WM_SYSCOMMAND` windows message to the `WMSysCommand()` method we have declared and defines the message structure to be `TWMSysCommand`.

In Delphi we define a message handler for the `WM_SYSCOMMAND` message for our form as follows (note we cannot override the handler from an ancestor as its not visible and not virtual):

```
type
   TfrmSystemMenu = class(TForm)
     ...
   private
     ...
   protected
     Procedure AddOptionsMenu();
     Procedure WMSysCommand(var Message : TWMSysCommand); Message WM_SYSCOMMAND;
   public
   end;
```

Next we define the implementation of these methods. For C++ it is as follows:

```
void __fastcall TfrmExpertManager::WMSysCommand(TWMSysCommand &Message ) {
    switch (Message.CmdType) {
     case iMenuID:
       ...
       // Do something here like show a dialogue :-)
       ...
       break;
   }
   TForm::Dispatch(&Message);
 };
```

I wanted to call the inherited method to ensure we didn't break the message handling of the ancestor classes but the method is not visible and therefore you cannot call it so instead I found that dispatching the message to the ancestor worked.

In Delphi is much easier as follows:

```
procedure TfrmSystemMenu.WMSysCommand(var Message: TWMSysCommand);
 begin
   inherited;
   case Message.CmdType of
     iMenuID: ShowMessage('Hello');
   end;
 end;
```

Here we can call the inherited method to ensure the message handling is not broken.

So is that all there is to it? Well no quite. See the reason I wanted the menu was to allow the user to change VCL themes in the application and that process of recreating the application will delete our menu. Bother!

Fixing the Menu After a Theme Change

So the following fix is only in C++ but I think you can translate it to Delphi. First you need a timer on the main form set to disabled and an interval of 1 second as we will need this to trigger the re-applying of the meun item. Why? Well, when you call `TThemeManager.TrySetStyle()` from within the message handler it does nothing as (I think) it sets a pending message to rebuild the application style, so we need to handle this after the new style is applied.

We trigger the timer in the above menu handler as follows:

```
void __fastcall TfrmExpertManager::WMSysCommand(TWMSysCommand &Message) {
    switch (Message.CmdType) {
     case iMenuID:
       FOptions.FVCLTheme = StyleServices()->Name;
       if (TfrmExpertOptions::Execute(FOptions)) {
         if (FOptions.FVCLTheme  = StyleServices()->Name) {
           TStyleManager::TrySetStyle(FOptions.FVCLTheme);
           tmSystemMenu->Enabled = true; // Trigger
         }
         ...
       }
       break;
   }
   TForm::Dispatch(&Message);
 };
```

The timer event handler is straightforward as follows:

```
void __fastcall TfrmExpertManager::SystemMenuTimerEvent(TObject* Sender) {
   tmSystemMenu->;Enabled = false;
```

```
    AddOptionsMenu();
  };
```

I hope that's of use to you. I certainly will use it in some of my application that I want to keep minimalistic.

regards

Dave.

Category: Applications C++ Builder Delphi Expert Manager RAD Studio