LDialog unit and how to use it

Problems in 16 bit applications built with Delphi 1.0x and VCL:

1. dialogs get under main form when switched to main form using taskbar in Win95
2. dialogs get under main form in single instance application when dialog is popped up in first instance and user tries to run another instance (he gets back first one with dialog under main form)
3. application with "always on top" menu. When this is marked on, application's main form and any dialog on top of main form should stay always on top of desktop. The problem is, any dialog gets always under main form and application looses "stay on top" ability
4. in Win95 each form creates its own icon visible in taskbar instead of one icon per application as other "real" applications do.

You can solve this problems by using LDialogs unit instead of standard VCL's Dialogs unit. LDialogs unit was built by me, Lester Kovac by modifying Dialogs unit's source code. It is the freeware with one condition: you must notify me at ag623@freenet.carleton.ca that you are using it. I would also appreciate any comments and suggestion. You can check also my web page for other "neat" stuff: http://www.cyberus.ca/~lkovac/lester.htm

Following is the description of LDialogs' use.

LDialogs unit declares also two new functions you can use.

```
function GetWindowOnTop(InstHandle: THandle): HWnd;
procedure CheckHelp(var Msg: TMessage);
```

First of all you have to rename each use of Dialogs in your source code (units and DPR) to LDialogs. There MUST NOT be even one usage of Dialogs in your application at all. Note that Delphi has some "bad" feature that whenever it starts or whenever you save file in IDE it tries to add you Dialogs unit again and again. The best thing to prevent it is to cheat Delphi IDE so that it will think you have Dialogs included but you actually have not. Here is the trick:

```
uses ... Other units ... , LDialogs  {$IFDEF NEVER} , Dialogs {$ENDIF} ;
```

NEVER is never defined, so Dialogs is never linked in, but Delphi IDE think you have it inside (got cheated by {$IFDEF} statement) so it won't add it for you unexpectedly. Even better, if you do it:

```
uses ... Other units ... , LDialogs  {$IFDEF NEVER} , Dialogs, BlahBlah {$ENDIF} ;
```

and, obviously, BlahBlah unit doesn't exist, you can make sure NEVER is never defined. If it was, compiler would yell at you that it cannot find it.

All your forms (except the main one) must override the CreateParams method so that it sets correctly it parent's handle. As example:

```
TForm2 = class(TForm)
    whatever...
private
    { Private-Declarations }
    procedure CreateParams(var Params: TCreateParams); override;
    ...
public
    { Public-Declarations }
    ...
```

```
end;
```

and in implementation:

```
procedure TForm2.CreateParams(var Params: TCreateParams);
begin
    inherited CreateParams(Params);
    Params.WndParent := Form1.Handle; { or whatever the parent's handle
   is... }
end;
```

Or even better, especially for the the dialogs (forms) which are displayed in one case on top on one form, in another case on another.

```
procedure TForm2.CreateParams(var Params: TCreateParams);
begin
    inherited CreateParams(Params);
    Params.WndParent := GetWindowOnTop(hInstance);
end;
```

Forms which are using GetWindowOnTop within CreateParams must be created only when they are needed, not in DPR file (just before they are to be displayed) and you must destroy them afterwards.. Here is the example:

```
procedure TForm2.Button2Click(Sender: TObject);
begin
    Form3 := TForm3.Create(Self);
    Form3.ShowModal;
    Form3.Free;
end;
```

To solve Win95 taskbar issue (problem 1) declare and add to your main form this method:

```
Procedure TMainForm.ApplicationActivate(ASender: TObject);
begin
    BringWindowToTop(GetWindowOnTop(hInstance));
end;
```

and anywhere inside FormCreate you have to do:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
    ...
    Application.OnActivate := ApplicationActivate;
    ...
end;
```

If you want to create single instance application, change body of your project (DPR) file:

```
if hPrevInst = 0 then
begin
    { do whatever was there before }
    Randomize;
    Application.CreateForm(TMainForm, MainForm);
    ...
    Application.Run;
end
```

```
else
begin
    PrevInstWnd := GetWindowOnTop(hPrevInst);
    if IsIconic(PrevInstWnd) then
        ShowWindow(PrevInstWnd, CmdShow)
    else
        BringWindowToTop(PrevInstWnd);
end;
```

Third problem ("Always on Top" application) requires to declare, implement and properly use procedure ManageStayOnTop (well, you can call it as you wish). In following example the status (to stay on top or not) is determined by menu item check mark:

```
procedure TMainForm.ManageStayOnTop;
var
    WndInsertAfter: HWnd;
begin
    if AlwaysOnTop1.Checked then
        WndInsertAfter := HWND_TOPMOST
    else
        WndInsertAfter := HWND_NOTOPMOST;
    SetWindowPos(Application.Handle, WndInsertAfter, 0, 0, 0, 0,
    SWP_NOMOVE or SWP_NOSIZE or SWP_NOACTIVATE);
    SetWindowPos(Handle, WndInsertAfter, 0, 0, 0, 0, SWP_NOMOVE or
    SWP_NOSIZE or SWP_NOACTIVATE);
end;
```

In order to work properly, ManageStayOnTop should be called when application looses focus and whenever "Stay on top status changes". So we ahve to declare and implement this method:

```
procedure TMainForm.ApplicationDeactivate(Sender: TObject);
begin
    if not Application.Terminated then
        ManageStayOnTop;
end;
```

And we have to set it to Application.OnDeactivate :

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
    ...
    Application.OnDeactivate := ApplicationDeactivate;
    ...
end;
```

And we have to call it on event which changes "Stay On Top" status. In this case, it is menu item click:

```
procedure TMainForm.AlwaysOnTop1Click(Sender: TObject);
begin
    AlwaysOnTop1.Checked := not AlwaysOnTop1.Checked;
    ManageStayOnTop;
end;
```

The last this you should know: what is implemented in LDialogs breaks F1 or Help button click in Common dialogs (File Open, File Save, etc.). In order to fix all properly I would need change much more than just one unit. If you don't need to implement help in common dialogs, just don't do anything. If you need it, the workaround is not difficult. You have to override WndProc for each window which hosts (i.e. is the parent of) any common dialog and call CheckHelp from LDialogs there:

```
procedure TForm1.WndProc(var Message: TMessage);
begin
    CheckHelp(Message);
    inherited WndProc(Message);
end;
```

And that's about it. Enjoy !!

This is help file which is used in test example where I link help topics to test application. Here are links used in example:

Dialog 1

Dialog 2

Dialog 3

File Open Dialog

File Save Dialog

Dialog 1 help

See also:

- LDialog units and how to use it

Dialog 2 help

See also:

- LDialog units and how to use it

Dialog 3 help

See also:

- LDialog units and how to use it

File Open Dialog help

See also:

- LDialog units and how to use it

File Save Dialog help

See also:

- LDialog units and how to use it