Dave's Development Blog
Software Development using Borland / Codegear / Embarcadero RAD Studio

Chapter 2: A simple custom menu (AutoSave)

By David | August 13, 2009 0 Comment

I thought that this time I would give you something useful for a change, so while implementing a simple custom menu we'll create a wizard that provides an auto save feature for the IDE.

With this second chapter I'm changing tact and instead of screen shots I'll include the source code (syntax highlighted). This means you can copy and paste the code easily. There will still be a zip file at the end of each chapter for the entire source code to the project.

First we need to to do some groundwork to create a form for editing the auto save options. I'm assuming here that you are familiar with forms so I'm just doing to give you the code. First the PAS file and then the DFM so you can paste the information into your IDE and we can get on with the interesting stuff. Either using the projects we created last time or copies of those projects add a new form to the DLL project and replace all the code with the following (REMEMEBER: if you create a copy of the projects, changes the GetIDString and GetName methods to reflect a different wizard):

```
unit OptionsForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ComCtrls;

type
  TfrmOptions = class(TForm)
    lblAutoSaveInterval: TLabel;
    edtAutosaveInterval: TEdit;
    udAutoSaveInterval: TUpDown;
    cbxPrompt: TCheckBox;
    btnOK: TBitBtn;
    btnCancel: TBitBtn;
  private
    { Private declarations }
  public
    { Public declarations }
```

```
    Class Function Execute(var iInterval : Integer; var boolPrompt : Boolean) : Boolean;
  end;

implementation

{$R *.DFM}

{ TfrmAutoSaveOptions }

class Function TfrmOptions.Execute(var iInterval: Integer;
  var boolPrompt: Boolean) : Boolean;

begin
  Result := False;
  With TfrmOptions.Create(Nil) Do
    Try
      udAutoSaveInterval.Position := iInterval;
      cbxPrompt.Checked := boolPrompt;
      If ShowModal = mrOK Then
        Begin
          Result := True;
          iInterval := udAutoSaveInterval.Position;
          boolPrompt := cbxPrompt.Checked;
        End;
    Finally
      Free;
    End;
end;

end.
```

Next view the form and right click on the form and selected “View as Text” and replace all the code with the following:

```
object frmOptions: TfrmOptions
  Left = 443
  Top = 427
  BorderStyle = bsDialog
  Caption = 'Auto Save Options'
  ClientHeight = 64
  ClientWidth = 241
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
```

```
  object lblAutoSaveInterval: TLabel
    Left = 8
    Top = 12
    Width = 88
    Height = 13
    Caption = 'Auto Save &Interval'
    FocusControl = edtAutosaveInterval
  end
  object edtAutosaveInterval: TEdit
    Left = 104
    Top = 8
    Width = 41
    Height = 21
    TabOrder = 0
    Text = '60'
  end
  object udAutoSaveInterval: TUpDown
    Left = 145
    Top = 8
    Width = 15
    Height = 21
    Associate = edtAutosaveInterval
    Min = 60
    Max = 3600
    Position = 60
    TabOrder = 1
  end
  object cbxPrompt: TCheckBox
    Left = 8
    Top = 36
    Width = 97
    Height = 17
    Caption = '&Prompt'
    TabOrder = 2
  end
  object btnOK: TBitBtn
    Left = 164
    Top = 8
    Width = 75
    Height = 25
    TabOrder = 3
    Kind = bkOK
  end
  object btnCancel: TBitBtn
    Left = 164
    Top = 36
    Width = 75
    Height = 25
    TabOrder = 4
    Kind = bkCancel
  end
end
```

Now save the form in the Source directory as OptionsForm.pas.

We can now get on with the fun bits of this chapter. First we need to update the class declaration of the wizard – we will not be needing the IOTAMenuWizard interface so this can be removed and the GetMenuText() method deleted.

```
TBlogOTAExampleWizard = Class(TInterfacedObject, IOTAWizard)
Public
  FTimer       : TTimer;                  // New
  FCounter     : Integer;                 // New
  FAutoSaveInt : Integer;                 // New
  FPrompt      : Boolean;                 // New
  FMenuItem    : TMenuItem;               // New
  FINIFileName : String;                  // New
  Procedure SaveModifiedFiles;            // New
Protected
  procedure Execute;
  function  GetIDString: string;
  function  GetName: string;
  function  GetState: TWizardState;
  procedure AfterSave;
  procedure BeforeSave;
  procedure Destroyed;
  procedure Modified;
  // function  GetMenuText: string;       // Deleted
  Procedure TimerEvent(Sender : TObject); // New
  Procedure MenuClick(Sender : TObject);  // New
  Procedure LoadSettings;                 // New
  Procedure SaveSettings;                 // New
Public
  Constructor Create;                     // New
  Destructor Destroy; Override;           // New
End;
```

Next we need to update the uses clause in the implementation section to provide access to other modules that will be required. I've take this opportunity to rename the wizard index variable so that it's clear what it refers to:

```
Uses
   // Delete Dialogs and add Windows, SysUtils, OptionsForm and IniFiles
  Forms, Windows, SysUtils, OptionsForm, IniFiles;

Var
  (** A private variable to hold the index returned by AddWizard which is needed
      by RemoveWizard. **)
  iWizardIndex : Integer = 0; // Renamed
```

Next we need to code the constructor. We need to initialise our fields, start the timer and create the menu as follows:

```
constructor TBlogOTAExampleWizard.Create;

var
  NTAS: INTAServices;
  mmiViewMenu: TMenuItem;
  mmiFirstDivider: TMenuItem;
  iSize : DWORD;

begin
  FMenuItem := Nil;
  FCounter := 0;
  FAutoSaveInt := 300; // Default 300 seconds (5 minutes)
  FPrompt := True;     // Default to True
  // Create INI file same as add module + '.INI'
  SetLength(FINIFileName, MAX_PATH);
  iSize := MAX_PATH;
  iSize := GetModuleFileName(hInstance, PChar(FINIFileName), iSize);
  SetLength(FINIFileName, iSize);
  FINIFileName := ChangeFileExt(FINIFileName, '.INI');
  LoadSettings;
  FTimer := TTimer.Create(Nil);
  FTimer.Interval := 1000; // 1 second
  FTimer.OnTimer := TimerEvent;
  FTimer.Enabled := True;
  NTAS := (BorlandIDEServices As INTAServices);
  If (NTAS <> Nil) And (NTAS.MainMenu <> Nil) Then
    Begin
      mmiViewMenu := NTAS.MainMenu.Items.Find('View');
      If mmiViewMenu <> Nil Then
        Begin
          //: @bug Menu not fully build at this point.
          mmiFirstDivider := mmiViewMenu.Find('-');
          If mmiFirstDivider <> Nil Then
            Begin
              FMenuItem := TMenuItem.Create(mmiViewMenu);
              FMenuItem.Caption := '&Auto Save Options...';
              FMenuItem.OnClick := MenuClick;
              mmiViewMenu.Insert(mmiFirstDivider.MenuIndex, FMenuItem);
            End;
        End;
    End;
end;
```

You will note that I've marked the menu creation code with a bug comment. What I found was that loading this wizard as a DLL loads the code much earlier in the IDE start-up process than loading it as a package. The consequence of this is that not all the IDE menus have been fully loaded. Originally, I was looking for the "Window List" menu item and inserting this new menu below it. I've copped out here and found the first separator in the menu and inserted the new menu above it. I will address this problems along with keyboard short cuts for menus in the next instalment. This only affects finding an IDE menu to reference against – creating your own top level menu would not be affected. I'll do this in a later chapter.

There's something else of interest in this code as well. I gave up using the windows registry some time ago as it can't be backed up such that you can restore your settings – so I elected to move back to the old fashioned INI file. Although I use slightly different code in my own applications (places the users name and computer name in the INI file name) this is essentially what I do. I use the Win32 API GetModuleFileName and pass it the hInstance global variable. What this means is that for DLLs and BPLs I get the name of the DLL, but for EXE I get the EXE name. If you were to use ParamStr(0) in the IDE you would get the Delphi / RAD Studio EXE name.

Next we need to code the destructor to ensure we free all the memory we've used as follows:

```
destructor TBlogOTAExampleWizard.Destroy;
begin
  SaveSettings;
  FMenuItem.Free;
  FTimer.Free;
  Inherited Destroy;
end;
```

You will notice that I call FMenuItem.Free even though it might not have been initialised (i.e. if the menu position was not found). This is in fact absolutely fine. Free is a class method and therefore can be called on a NIL reference, hence why I ensure its initialised to NIL in the constructor. One of the Borland / CodeGear guys wrote about this a couple of years ago and explain why this would work – I just don't think its widely known.

The next thing to do is implement the loading and saving code for the wizard's settings as follows:

```
procedure TBlogOTAExampleWizard.LoadSettings;

begin
  With TIniFile.Create(FINIFileName) Do
    Try
      FAutoSaveInt := ReadInteger('Setup', 'AutoSaveInt', FAutoSaveInt);
      FPrompt := ReadBool('Setup', 'Prompt', FPrompt);
    Finally
      Free;
    End;
end;

procedure TBlogOTAExampleWizard.SaveSettings;

begin
  With TIniFile.Create(FINIFileName) Do
    Try
      WriteInteger('Setup', 'AutoSaveInt', FAutoSaveInt);
      WriteBool('Setup', 'Prompt', FPrompt);
    Finally
```

```
      Free;
    End;
end;
```

We can expand these routines later to load and save more settings.

Next we'll code the timer event handler. We simply call the SaveModifiedFiles routine when the counter gets larger than the interval and reset the counter at the same time.

```
procedure TBlogOTAExampleWizard.TimerEvent(Sender: TObject);

begin
  Inc(FCounter);
  If FCounter >= FAutoSaveInt Then
    Begin
      FCounter := 0;
      SaveModifiedFiles;
    End;
end;
```

Next we'll code the MenuClick event handler passing our two fields as parameters so that the options dialogue at the start of this chapter can modfied the values.

```
procedure TBlogOTAExampleWizard.MenuClick(Sender: TObject);

begin
  If TfrmOptions.Execute(FAutoSaveInt, FPrompt) Then
    SaveSettings; // Not really required as is called in destructor.
end;
```

Finally we come to the interesting bit, saving the modified files in the IDE.

```
procedure TBlogOTAExampleWizard.SaveModifiedFiles;

Var
  Iterator : IOTAEditBufferIterator;
  i : Integer;

begin
  If (BorlandIDEServices As IOTAEditorServices).GetEditBufferIterator(Iterator) Then
    Begin
      For i := 0 To Iterator.Count - 1 Do
        If Iterator.EditBuffers[i].IsModified Then
```

```
            Iterator.EditBuffers[i].Module.Save(False, Not FPrompt);
        End;
    end;
```

Here we ask the IDE for a Edit Buffer Iterator and use that iterator to check each file in the editor to see if it has been modified and if it has then save the modifications.

Well I hope this is straight forward. Here is the zip file of the source.

Category: Open Tools API Tags: Borland, CodeGear, Delphi, Experts, IOTAEditBufferIterator, IOTAEditorServices, IOTAWizard, Menus, RAD Studio