Dave's Development Blog

Software Development using Borland / Codegear / Embarcadero RAD Studio

Adding Menu Items to the IDE Editor’s Context Menu

By David | November 25, 2019 0 Comment

I decided not so long ago to create a new IDE plug-in to consolidate a few separate tools that are related to debugging. While doing this I found (as others had) that changes in the way the IDE handles the context menu in the editor stopped my code being able to insert context menu items.

I believe this is because the IDE context menu is now dynamic and not static and gets created every time it’s required rather than once at the IDE start-up.

So the following is how I solved the problem and it starts with a Timer (no eggs were hurt in the experiment). I generally create DLL plug-ins and I found that the Editor and its context menu are not available when my plug-in is initialised so I need to wait until they are available and then do something. Below is the timer code (note, all this code is in my plug-in wizard).

```
Procedure TDDTWizard.MenuInstallerTimer(Sender: TObject);

Begin
  HookEditorPopupMenu;
End;
```

Nothing outrageous here, it just calls a method to hook the editor pop-up menu, the name should hint to you what I’m about to do but no peeking.

The timer is simply started from the constructor for the wizard as follows:

```
Constructor TDDTWizard.Create;

Const
   iTimerInterval = 1000;

Begin
  Inherited Create;
  ...
  FEditorPopupMethod.Data := Nil;
```

```
  FMenuInstalled := False;
  FMenuTimer := TTimer.Create(Nil);
  FMenuTimer.Interval := iTimerInterval;
  FMenuTimer.OnTimer := MenuInstallerTimer;
  FMenuTimer.Enabled := True;
End;
```

Again nothing unusual here except for the initialisation of the `FEditorPopupMethod.Data` member. `FEditorPopupMethod` is declared as a `TMethod` and we are going to use it to hold a reference to an existing editor pop-up menu event handlers.

So now for the interesting bit – what's in the `HookEditorPopupMenu` method.

```
Procedure TDDTWizard.HookEditorPopupMenu;

Var
  EditorPopupMenu : TPopupActionBar;

Begin
  EditorPopupMenu := FindEditorPopup;
  If Assigned(EditorPopupMenu) Then
    Begin
      FImageIndex := AddImageToList(EditorPopupMenu.Images);
      If Assigned(EditorPopupMenu.OnPopup) Then
        Begin
          FEditorPopupMethod := TMethod(EditorPopupMenu.OnPopup);
          EditorPopupMenu.OnPopup := DebuggingToolsPopupEvent;
        End Else
          EditorPopupMenu.OnPopup := DebuggingToolsPopupEvent;
      FMenuTimer.Enabled := False;
    End;
End;
```

The above code attempts to find the Editor Popup Menu using a custom function `FindEditorPopup()` which we'll talk about in a moment. If it gets a valid reference then an image is installed into the Popup Menu's image list and then one or two things can happen.

If the Editor Popup Menu already has an event handler (either because the IDE's set one or another plug-in did) then we need to store this and install our own else we just install our own.

Once we've done this then we can disabled the timer as we only need to hook this once.

So what does `FindEditorPopup()` do? Let's see…

```
Function  TDDTWizard.FindEditorPopup : TPopupActionBar;

Const
  strEditorLocalMenuComponentName = 'EditorLocalMenu';
```

```
Var
  EditorForm: TForm;

Begin
  Result := Nil;
  EditorForm := FindEditWindow;
  If Assigned(EditorForm) Then
    Begin
      Result := FindComponent(
        EditorForm,
        strEditorLocalMenuComponentName,
        TPopupActionBar
      ) As TPopupActionBar;
    End;
End;
```

This function delegates to two other functions to find the Editor window and the Popup Menu component and they are implemented as follows:

```
Function TDDTWizard.FindEditWindow : TForm;

Const
  strTEditWindowClassName = 'TEditWindow';

Var
  iForm: Integer;

Begin
  Result := Nil;
  For iForm := 0 To Screen.FormCount - 1 Do
    If CompareText(Screen.Forms[iForm].ClassName, strTEditWindowClassName) = 0 Then
      Begin
        Result := Screen.Forms[iForm];
        Break;
      End;
End;
```

This method searches for the `TEditWindow` class name in the IDE's list of forms and returns its reference if found.

```
Function TDDTWizard.FindComponent(Const OwnerComponent : TComponent; Const strName :
String; Const ClsType : TClass) : TComponent;

Var
```

```
  iComponent: Integer;

Begin
  Result := Nil;
  For iComponent := 0 To OwnerComponent.ComponentCount - 1 Do
    If CompareText(OwnerComponent.Components[iComponent].Name, strName) = 0 Then
      If OwnerComponent.Components[iComponent] Is ClsType Then
        Begin
          Result := OwnerComponent.Components[iComponent];
          Break;
        End;
End;
```

The above searches the given component for the given component name with the given class type, i.e. we call this looking for the Popup Menu.

While we're at it we should say what happens in the `AddImageToList` method. Here we extract a bitmap from the DLL's resources and add it to the image list associated with the Popup Menu.

```
Function TDDTWizard.AddImageToList(Const ImageList : TCustomImageList): Integer;

Const
  strImageName = 'DDTMenuBitMap16x16';

Var
  BM : VCL.Graphics.TBitMap;

Begin
  Result := -1;
  If FindResource(hInstance, strImageName, RT_BITMAP) > 0 Then
    Begin
      BM := VCL.Graphics.TBitMap.Create;
      Try
        BM.LoadFromResourceName(hInstance, strImageName);
        Result := ImageList.AddMasked(BM, clLime);
      Finally
        BM.Free;
      End;
    End;
End;
```

So next we need to look at what the event handler we've installed does to actually add out menu items.

```
Procedure TDDTWizard.DebuggingToolsPopupEvent(Sender: TObject);

ResourceString
```

```
  strDebuggingTools = 'Debugging Tools';
  strAddBreakpoint = 'Add Breakpoint';
  strDebugWithCodeSiteCaption = 'Debug &with CodeSite';

Var
  NotifyEvent : TNotifyEvent;
  EditorPopupMenu: TPopupActionBar;
  MI: TMenuItem;

Begin
  If Assigned(FEditorPopupMethod.Data) Then
    Begin
      NotifyEvent := TNotifyEvent(FEditorPopupMethod);
      NotifyEvent(Sender);
    End;
  EditorPopupMenu := FindEditorPopup;
  If Assigned(EditorPopupMenu) Then
    Begin
      If Assigned(FDebuggingToolsMenu) Then
        FDebuggingToolsMenu.Free;
      // Create Main Menu Item
      FDebuggingToolsMenu := TMenuItem.Create(EditorPopupMenu);
      FDebuggingToolsMenu.Caption := strDebuggingTools;
      //FDebuggingToolsMenu.OnClick := DebugWithCodeSite;
      FDebuggingToolsMenu.ImageIndex := FImageIndex;
      EditorPopupMenu.Items.Add(FDebuggingToolsMenu);
      // Create Add Breapoint
      MI := TMenuItem.Create(FDebuggingToolsMenu);
      MI.Caption := strAddBreakpoint;
      MI.OnClick := AddBreakpoint;
      MI.ImageIndex := FImageIndex;
      FDebuggingToolsMenu.Add(MI);
      // Create Debug with CodeSite
      MI := TMenuItem.Create(FDebuggingToolsMenu);
      MI.Caption := strDebugWithCodeSiteCaption;
      MI.OnClick := DebugWithCodeSite;
      MI.ImageIndex := FImageIndex;
      FDebuggingToolsMenu.Add(MI);
    End;
End;
```

This first thing we do is see if we had a previous event handler for the Popup Menu and if so we call it to ensure that either the IDE's code or another plug-in's code is run.

Once we have done this we get a reference to the Editor Popup Menu and add three menu items, the first a

parent for the other two cascading menus and we hook these to `TNotifyEvent` methods which do the actual work we require.

That's it, all done, or is it…

Well for a DLL yes as you would never get a situation where the Editor Popup Menu is called when you DLL is not in memory any more however this is not true for a BPL based plug-in so we need to reverse our hook. This starts in the Wizard's destructor as follows:

```
Destructor TDDTWizard.Destroy;

Begin
  UnhookEditorPopupMenu;
  ...
  FMenuTimer.Free;
  Inherited Destroy;
End;
```

This in turn calls the code to unhook the menu event as follows:

```
Procedure TDDTWizard.UnhookEditorPopupMenu;

Var
  EditorPopupMenu : TPopupActionBar;

Begin
  {$IFDEF DEBUG} CodeSite.TraceMethod(Self, 'UnhookEditorPopupMenu', tmoTiming);
{$ENDIF DEBUG}
  EditorPopupMenu := FindEditorPopup;
  If Assigned(EditorPopupMenu) And Assigned(EditorPopupMenu.OnPopup) Then
    EditorPopupMenu.OnPopup := TNotifyEvent(FEditorPopupMethod);
End;
```

This simply finds the Editor Popup Menu and re-assigns the event handler we stored in the `FEditorPopupMenu` reference.

Knowing there are changes in how this is implemented in different IDEs I've test this code back to XE8 and all seems to work as expected.

Enjoy D.

Category: Debugging Tools Delphi Open Tools API RAD Studio