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

Notify Me of Everything… – Part 2.1 (Rename Fix)

By David | October 31, 2017 0 Commen

Overview

This is an update to the second article Notify Me of Everything… – Part 2. While I was fixing some performance bugs in the plug-in I thought it would be good to fix the issue of renamed modules at the same time.

The Fix

One of the things I've been doing recently when revisiting code and refactoring it, is to use interfaces to decouple the code. So I looked at the code I had written for the management of the notifier indexes against filenames and decided that that code should be separated out into a class of its own so that it can be reused. With that in mind I delcared the following interface for the class.

```
Type
  IDINModuleNotifierList = Interface
  ['{60E0D688-F529-4798-A06C-C283F800B7FE}']
    Procedure Add(Const strFileName : String; Const iIndex : Integer);
    Function  Remove(Const strFileName: String): Integer;
    Procedure Rename(Const strOldFileName, strNewFileName : String);
  End;
```

With the above interface I can then create a new class which encapsulates the generic collection I was using before. I've also added to the class the record type which described the data to be stored as the methods of the interface expose all the data required. I have also moved across the Find method. Although it uses a sequential search, the dataset should be small and its not called too often to require a binary search function. The below also allows me to have one of these collection for Modules and another for Projects. This also provides me a useful class to reuse later on when I try and implement other module based notifiers.

```
Type
  TDINModuleNotifierList = Class(TInterfacedObject, IDINModuleNotifierList)
  Strict Private
    Type
      TModuleNotifierRec = Record
      Strict Private
        FFileName      : String;
        FNotifierIndex : Integer;
      Public
        Constructor Create(Const strFileName : String; Const iIndex : Integer);
        Property FileName : String Read FFileName Write FFileName;
        Property NotifierIndex : Integer Read FNotifierIndex;
      End;
  Strict Private
    FModuleNotifierList : TList<TModuleNotifierRec>;
  {$IFDEF D2010} Strict {$ENDIF} Protected
    Procedure Add(Const strFileName : String; Const iIndex : Integer);
    Function  Remove(Const strFileName: String): Integer;
    Procedure Rename(Const strOldFileName: String; Const strNewFileName: String);
    Function Find(Const strFileName : String; Var iIndex : Integer) : Boolean;
  Public
    Constructor Create;
```

```
    Destructor Destroy; Override;
  End;
```

The record has the following simple constructor just to simplify the creaton and initialise the record.

```
Constructor TDINModuleNotifierList.TModuleNotifierRec.Create(Const strFileName: String;
  Const iIndex: Integer);

Begin
  FFileName := strFileName;
  FNotifierIndex := iIndex;
End;
```

The Add method of the class constructs a new record with the filename of the module and the notifier index and adds it to the end of the generic collection.

```
Procedure TDINModuleNotifierList.Add(Const strFileName: String; Const iIndex: Integer);

Begin
  FModuleNotifierList.Add(TModuleNotifierRec.Create(strFileName, iIndex));
End;
```

The constructor for the class simply creates the generic collection to hold the filename index pairs.

```
Constructor TDINModuleNotifierList.Create;

Begin
  FModuleNotifierList := TList<TModuleNotifierRec>.Create;
End;
```

The destructor deletes any remaining record in the collection before freeing the memory used by the collection.

```
Destructor TDINModuleNotifierList.Destroy;

Var
  iModule : Integer;

Begin
  For iModule := FModuleNotifierList.Count - 1 DownTo 0 Do
    Begin
      FModuleNotifierList.Delete(iModule);
      //: @note Cannot remove any left over notifiers here as the module
      //:       is most likely closed at ths point however there should be any anyway.
    End;
  FModuleNotifierList.Free;
  Inherited Destroy;
End;
```

The Find method, used to search for a record with a specific filename, uses a simple linear search. The collections shouldn't be too large and the method should only be called for closing modules and renaming modules, so infrequently.

```
Function TDINModuleNotifierList.Find(Const strFileName: String; Var iIndex: Integer): Boolean;

Var
  iModNotIdx : Integer;
  R: TModuleNotifierRec;

Begin
  Result := False;
  iIndex := -1;
  For iModNotIdx := 0 To FModuleNotifierList.Count - 1 Do
    Begin
      R := FModuleNotifierList.Items[iModNotIdx];
```

```
      If CompareText(R.FileName, strFileName) = 0 Then
        Begin
          iIndex := iModNotIdx;
          Result := True;
          Break;
        End;
    End;
End;
```

The Remove method searches for the corresponding record and if found deletes the record from the collection and returns the notifier index which should be used to remove the notifier from the IDE.

```
Function TDINModuleNotifierList.Remove(Const strFileName: String): Integer;

Var
  iModuleIndex: Integer;
  R : TModuleNotifierRec;

Begin
  Result := -1;
  If Find(strFileName, iModuleIndex) Then
    Begin
      R := FModuleNotifierList[iModuleIndex];
      Result := R.NotifierIndex;
      FModuleNotifierList.Delete(iModuleIndex);
    End;
End;
```

The Rename method is very similar to the remove method however after finding the corresponding record it updates the notifier filename.

```
Procedure TDINModuleNotifierList.Rename(Const strOldFileName, strNewFileName: String);

Var
  iIndex : Integer;
  R : TModuleNotifierRec;

Begin
  If Find(strOldFileName, iIndex) Then
    Begin
      R := FModuleNotifierList[iIndex];
      R.FileName := strNewFileName;
      FModuleNotifierList[iIndex] := R;
    End;
End;
```

In order for the module and project notifiers to be able to update the filename when they change they need a reference to the collection containing their filenames and notifier indexes. Here is where the interface comes into play and which is passed to the notifier in a modified constructor.

```
Type
  TDNModuleNotifier = Class(TDGHNotifierObject, IOTAModuleNotifier, IOTAModuleNotifier80,
    IOTAModuleNotifier90)
  Strict Private
    FModuleNotiferList: IDINModuleNotifierList;
  {$IFDEF D2010} Strict {$ENDIF} Protected
    Function CheckOverwrite: Boolean;
    Procedure ModuleRenamed(Const NewName: String);
    // IOTAModuleNotifier80
    Function AllowSave: Boolean;
    Function GetOverwriteFileNameCount: Integer;
    Function GetOverwriteFileName(Index: Integer): String;
```

```
    Procedure SetSaveFileName(Const FileName: String);
    // IOTAModuleNotifier90
    Procedure BeforeRename(Const OldFileName, NewFileName: String);
    Procedure AfterRename(Const OldFileName, NewFileName: String);
    // General Properties
    Property RenameModule : IDINModuleNotifierList Read FModuleNotiferList;
  Public
    Constructor Create(Const strNotifier, strFileName: String;
      Const iNotification : TDGHIDENotification; Const RenameModule: IDINModuleNotifierList);
        Reintroduce; Overload;
  End;
```

The revised constructor is as follows:

```
Constructor TDNModuleNotifier.Create(Const strNotifier, strFileName: String;
  Const iNotification: TDGHIDENotification; Const RenameModule: IDINModuleNotifierList);

Begin
  Inherited Create(strNotifier, strFileName, iNotification);
  FModuleNotiferList := RenameModule;
End;
```

Finally back in the IDE notifier the original references to the generic collection are replaced with two references: one for the modules notifiers and another for the project notifiers.

```
Type
  TDGHNotificationsIDENotifier = Class(TDGHNotifierObject, IOTAIDENotifier,
    IOTAIDENotifier50, IOTAIDENotifier80)
  Strict Private
    FModuleNotifiers  : IDINModuleNotifierList;
    FProjectNotifiers : IDINModuleNotifierList;
  {$IFDEF D2010} Strict {$ENDIF} Protected
    // IOTAIDENotifier
    Procedure FileNotification(NotifyCode: TOTAFileNotification;
      Const FileName: String; Var Cancel: Boolean);
    // IOTAIDENotifier
    Procedure BeforeCompile(Const Project: IOTAProject; Var Cancel: Boolean); Overload;
    Procedure AfterCompile(Succeeded: Boolean); Overload;
    // IOTAIDENotifier50
    Procedure BeforeCompile(Const Project: IOTAProject; IsCodeInsight: Boolean;
      Var Cancel: Boolean); Overload;
    Procedure AfterCompile(Succeeded: Boolean; IsCodeInsight: Boolean); Overload;
    // IOTAIDENotifier80
    Procedure AfterCompile(Const Project: IOTAProject; Succeeded:
      Boolean; IsCodeInsight: Boolean); Overload;
  Public
    Constructor Create(Const strNotifier, strFileName : String;
      Const iNotification : TDGHIDENotification); Override;
    Destructor Destroy; Override;
  End;
```

The `FileNotification()` method is now a little more straightward as follows:

```
Procedure TDGHNotificationsIDENotifier.FileNotification(NotifyCode: TOTAFileNotification;
  Const FileName: String; Var Cancel: Boolean);

Const
  strNotifyCode : Array[Low(TOTAFileNotification)..High(TOTAFileNotification)] Of String = (
    'ofnFileOpening',
    'ofnFileOpened',
    'ofnFileClosing',
    'ofnDefaultDesktopLoad',
```

```
    'ofnDefaultDesktopSave',
    'ofnProjectDesktopLoad',
    'ofnProjectDesktopSave',
    'ofnPackageInstalled',
    'ofnPackageUninstalled',
    'ofnActiveProjectChanged' {$IFDEF DXE80},
    'ofnProjectOpenedFromTemplate' {$ENDIF}
  );

Var
  MS : IOTAModuleServices;
  M : IOTAModule;
  P : IOTAProject;
  MN : TDNModuleNotifier;
  C : IDINModuleNotifierList;
  iIndex : Integer;

Begin
  DoNotification(
    Format(
    '.FileNotification = NotifyCode: %s, FileName: %s, Cancel: %s',
      [
        strNotifyCode[NotifyCode],
        ExtractFileName(FileName),
        strBoolean[Cancel]
      ])
  );
  If Not Cancel And Supports(BorlandIDEServices, IOTAModuleServices, MS) Then
    Case NotifyCode Of
      ofnFileOpened:
        Begin
          M := MS.OpenModule(FileName);
          If Supports(M, IOTAProject, P) Then
            Begin
              MN := TDNProjectNotifier.Create('IOTAProjectNotifier', FileName, dinProjectNotifier,
                FProjectNotifiers);
              FProjectNotifiers.Add(FileName, M.AddNotifier(MN));
            End Else
            Begin
              MN := TDNModuleNotifier.Create('IOTAModuleNotifier', FileName, dinModuleNotifier,
                FModuleNotifiers);
              FModuleNotifiers.Add(FileName, M.AddNotifier(MN));
            End;
        End;
      ofnFileClosing:
        Begin
          M := MS.OpenModule(FileName);
          If Supports(M, IOTAProject, P) Then
            C := FProjectNotifiers
          Else
            C := FModuleNotifiers;
          iIndex := C.Remove(FileName);
          If iIndex > -1 Then
            M.RemoveNotifier(iIndex);
        End;
    End;
End;
```

Hopefully all of the the above is straightforward. The new code can be found with the updated plug-in on its page IDE Notifications.

Related posts:

1. Notify Me of Everything… – Part 2 (15.5)
2. Notify me of everything… – Part 1 (8.4)
3. Chapter 4: Key Bindings and Debugging Tools (6.4)
4. Chapter 4.1 – The Fix (6.2)
5. Chapter 11: Writing editor code (5.7)

Category: IDE Notifications Open Tools API RAD Studio