Dave's Development Blog

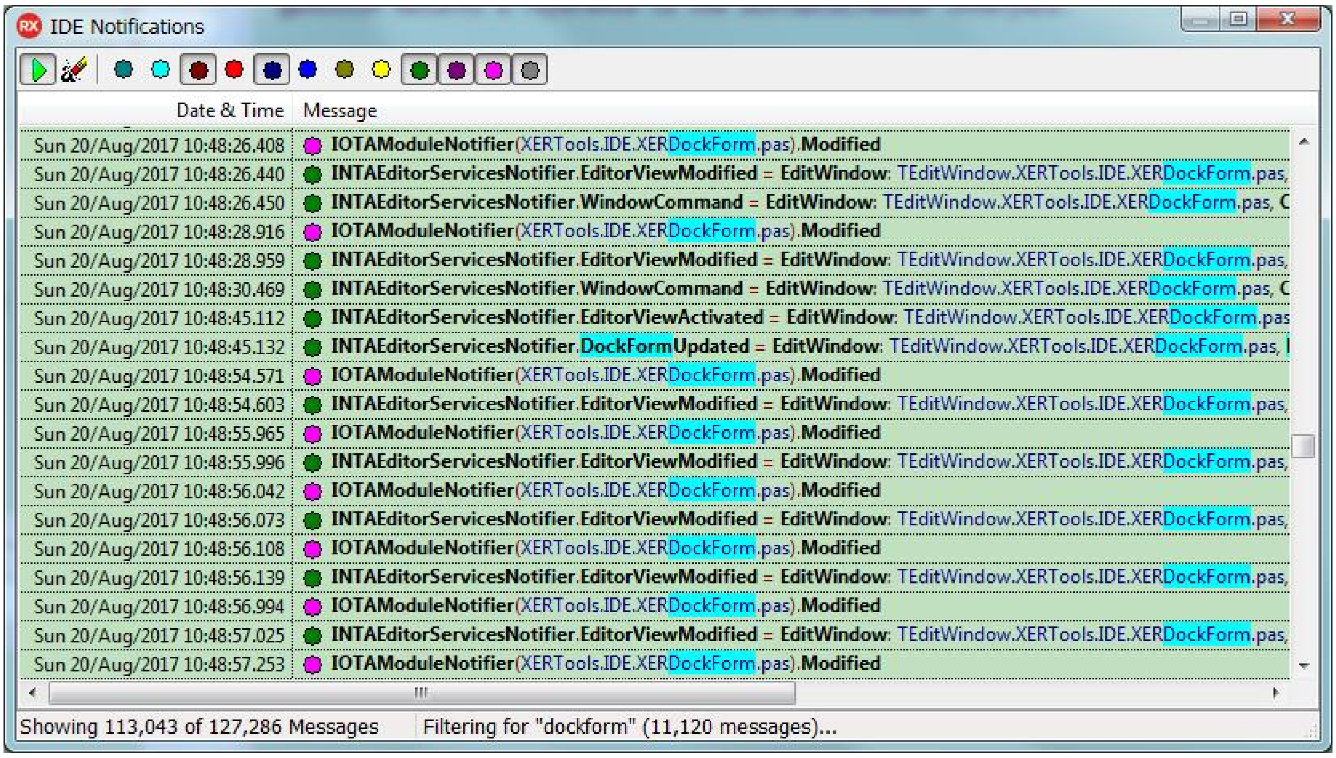
Software Development using Borland / Codegear / Embarcadero RAD Studio

Notify Me of Everything… – Part 2

By David | September 21, 2017 0 Commen

Overview

This is a second article on notifications in the RAD Studio IDE and in it I'll look at two slightly harder notifications to implement: Module and Project notifiers.



```
Date & Time                  Message
Sun 20/Aug/2017 10:48:26.408 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:26.440 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:26.450 ● INTAEditorServicesNotifier.WindowCommand = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas, C
Sun 20/Aug/2017 10:48:28.916 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:28.959 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:30.469 ● INTAEditorServicesNotifier.WindowCommand = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas, C
Sun 20/Aug/2017 10:48:45.112 ● INTAEditorServicesNotifier.EditorViewActivated = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas
Sun 20/Aug/2017 10:48:45.132 ● INTAEditorServicesNotifier.DockFormUpdated = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas, I
Sun 20/Aug/2017 10:48:54.571 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:54.603 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:55.965 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:55.996 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:56.042 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:56.073 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:56.108 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:56.139 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:56.994 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Sun 20/Aug/2017 10:48:57.025 ● INTAEditorServicesNotifier.EditorViewModified = EditWindow: TEditWindow.XERTools.IDE.XERDockForm.pas,
Sun 20/Aug/2017 10:48:57.253 ● IOTAModuleNotifier(XERTools.IDE.XERDockForm.pas).Modified
Showing 113,043 of 127,286 Messages    Filtering for "dockform" (11,120 messages)...
```

Module Notifier

The first notifier I'll go through is the module notifier. Note that I've derived it from my own base-notifier object which handlers the common `IOTANotifier` methods.

Interface

There are 3 interfaces to be implemented for this notifier: `IOTAModuleNotifer`, `IOTAModuleNotifier80` and `IOTAModuleNotifier90` as shown below.

```
Type
  TDNModuleNotifier = Class(TDGHNotifierObject, IOTAModuleNotifier, IOTAModuleNotifier80,
    IOTAModuleNotifier90)
  Strict Private
  {$IFDEF D2010} Strict {$ENDIF} Protected
    // IOTAModuleNotifier
    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);
  Public
  End;
```

You will notice that I've had to put an `IFDEF` around the `Strict` keyword for RAD Studio 2009 and below. I put the interface methods in a `Strict Protected` section so that I don't inadvertently use those methods from a class reference (cause its bad and causes reference counting problems). But it seems RAD Studio 2009 and below doesn't like a `Strict Protected` section but can handle a `Protected` section.

Implementation

I've broken down the implementations into their interfaces below.

IOTAModuleNotifier Interface Methods

The below method `ModuleRenamed` of the notifier is called when a module has been renamed.

```
Procedure TDNModuleNotifier.ModuleRenamed(Const NewName: String);

Begin
  DoNotification(
    Format(
    '80(%s).ModuleRenamed = NewName: %s',
      [
        ExtractFileName(FileName),
        ExtractFileName(NewName)
      ])
  );
  FileName := NewName;
End;
```

The method `CheckOverwrite` of the notifier is called before a Save As operation to check if any files that are read only file will be overwritten. I think here you should return `true` if any files you manage along with the module are read-only.

```
Function TDNModuleNotifier.CheckOverwrite: Boolean;

Begin
  Result := True;
  DoNotification(Format('(%s).CheckOverwrite = Result: True', [ExtractFileName(FileName)]));
End;
```

IOTAModuleNotifier80 Interface Methods

The method `AllowSave` of the notifier is called to check whether your notifier will allow the module to be saved. Return `true` to allow the module to be saved by the IDE else return `false` to prevent saving the module.

```
Function TDNModuleNotifier.AllowSave: Boolean;

Begain
  Result := True;
  DoNotification(Format('80(%s).AllowSave = Result: True', [ExtractFileName(FileName)]));
End;
```

The method `GetOverwriteFileName` of the notifier is called so that you can return a number of files (in addition to those managed by the IDE) that you want to manage along with the module.

```
Function TDNModuleNotifier.GetOverwriteFileName(Index: Integer): String;
```

```
Begin
  Result := '';
  DoNotification(Format('(%s).GetOverwriteFileName = Index: %d, Result: ''''', [
    ExtractFileName(FileName), Index]));
End;
```

The method `GetOverwriteFileNameCount` of the notifier is called so that you can return the number of files (in addition to those managed by the IDE) that you want to manage along with the module and specifically to be checked by the IDE during a Save As operation.

```
Function TDNModuleNotifier.GetOverwriteFileNameCount: Integer;

Begin
  Result := 0;
  DoNotification(Format('(%s).GetOverwriteFileNameCount = Result: 0', [ExtractFileName(FileName)]));
End;
```

The method `SetSaveFileName` of the notifier is called with the fully qualified filename that the user entered in the Save As dialogue. This name can then be used to determine all the resulting names.

```
Procedure TDNModuleNotifier.SetSaveFileName(Const FileName: String);

Begin
  DoNotification(
    Format(
    '80(%s).SetSaveFileName = FileName: %s',
      [
        ExtractFileName(FileName),
        ExtractFileName(FileName)
      ])
  );
End;
```

IOTAModuleNotifier90 Interface Methods

The method `AfterRename` of the notifier is called after a module has been renamed providing the old and new filenames.

```
Procedure TDNModuleNotifier.AfterRename(Const OldFileName, NewFileName: String);

Begin
  DoNotification(
    Format(
    '90(%s).AfterRename = OldFileName: %s, NewFileName: %s',
      [
        ExtractFileName(FileName),
        ExtractFileName(OldFileName),
        ExtractFileName(NewFileName)
      ])
  );
End;
```

The method `BeforeRename` of the notifier is called before a module is renamed providing the old and new file names.

```
Procedure TDNModuleNotifier.BeforeRename(Const OldFileName, NewFileName: String);

Begin
  DoNotification(
    Format(
    '90(%s).BeforeRename = OldFileName: %s, NewFileName: %s',
      [
        ExtractFileName(FileName),
        ExtractFileName(OldFileName),
        ExtractFileName(NewFileName)
      ])
  );
End;
```

Project Notifier

The second notifer I'm going to look at is related to the module notifier and is a project [module] notifier.

Interface

I've derived this notifier from the above module notifier as `IOTAProjectNotifier` is inherited from `IOTAModuleNotifier`.

```
TDNProjectNotifier = Class(TDNModuleNotifier, IOTAProjectNotifier)
  Strict Private
  {$IFDEF D2010} Strict {$ENDIF} Protected
    // IOTAProjectModule
    Procedure ModuleAdded(Const AFileName: String);
    Procedure ModuleRemoved(Const AFileName: String);
    Procedure ModuleRenamed(Const AOldFileName, ANewFileName: String); {$IFNDEF D2010} Overload; {$ENDIF}
  Public
  End;
```

You should notice from the above that there is an `IFDEF`ed `Override` directive for the `ModuleRenamed` method. This is required for RAD Studio 2009 and before to help the compiler resolve the reference. Later IDEs don't seem to require it.

Implementation

Below are some explanations for the methods.

IOTAProjectNotifier Interface

This method of the notifier is called when a module is added to a project or a project is added to a project group.

```
Procedure TDNProjectNotifier.ModuleAdded(Const AFileName: String);

Begin
  DoNotification(Format('(%s).ModuleAdded = AFileName: %s', [FileName,
    ExtractFileName(AFileName)]));
End;
```

This method of the notifier is called when a module is removed from a project or a project is removed from a project group.

```
Procedure TDNProjectNotifier.ModuleRemoved(Const AFileName: String);

Begin
  DoNotification(Format('(%s).ModuleRemoved = AFileName: %s', [FileName,
    ExtractFileName(AFileName)]));
End;
```

This method is called when a project or project group has its name changed.

```
Procedure TDNProjectNotifier.ModuleRenamed(Const AOldFileName, ANewFileName: String);

Begin
  DoNotification(Format('(%s).ModuleRenamed = AOldFileName: %s, ANewFileName: %s',
    [FileName, ExtractFileName(AOldFileName), ExtractFileName(ANewFileName)]));
  FileName := ANewFileName;
End;
```

Adding and Removing the Notifiers

Now for the hard part, finding a sensible place to add and remove the notifiers. I've chosen the `TDGHNotificationsIDENotifier` notifier as the best place to do this as the `FileNotification` method informs you when a module is opened and closed.

In order to track the notifiers that are added I needed a collection to hold the files names and their notifier indexes so I created a record for the information which can then be transformed into a collection using a generic `TList`. Below is the definition of the record.

```
Type
  TModNotRec = Record
  Strict Private
    FFileName      : String;
    FNotifierIndex : Integer;
    FNotifierType  : TDGHIDENotification;
  Public
    Constructor Create(Const strFileName : String; Const iIndex : Integer;
      Const eNotifierType : TDGHIDENotification);
    Property FileName : String Read FFileName;
    Property NotifierIndex : Integer Read FNotifierIndex;
    Property NotifierType : TDGHIDENotification Read FNotifierType;
  End;
```

I've then defined an intermediate type for the collection as shown below. This is not necessary, I've only done this as it highlighted a bug in my Browse and Doc It parser that needs to be fixed.

```
TModNotRecList = TList<TModNotRec>;
```

Below is the updated definition of the IDE notifier class with the collection field.

```
Type
  TDGHNotificationsIDENotifier = Class(TDGHNotifierObject, IOTAIDENotifier,
    IOTAIDENotifier50, IOTAIDENotifier80)
  Strict Private
    FModuleNotifierRefs : TModNotRecList;
  {$IFDEF D2010} Strict {$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;
    Function Find(Const strFileName : String; Var iIndex : Integer) : Boolean;
    Property ModuleNotifierRefs : TModNotRecList Read FModuleNotifierRefs;
  Public
    Constructor Create(Const strNotifier, strFileName : String;
      Const iNotification : TDGHIDENotification); Override;
    Destructor Destroy; Override;
  End;
```

We need to add a constructor to the class so that we can create the collection to store the module and project notifiers as shown below.

```
Constructor TDGHNotificationsIDENotifier.Create(Const strNotifier, strFileName : String;
  Const iNotification : TDGHIDENotification);

Begin
  Inherited Create(strNotifier, strFileName, iNotification);
  FModuleNotifierRefs := TModNotRecList.Create;
End;
```

We also need to free the collection in a destructor as shown below. I also thought that I could remove any project and module notifiers that were still around but I realised that they were probably gone from the IDE at this point so they cannot be removed. Why would there be any dangling notifiers? Well there is a bug in the code at present. The notifier index reference is added to the collection using the original file name of the module. If a module is renamed then the removal process which will be described below will not work. I have 2 choices to fix this: the first is an event handler passed to each module / project notifier so they can update the collection file name or I pass an interface which does the same thing. The interface method would be the better way of doing it however I need to ensure I don't repeat the mistake I recently made where one object as a reference to another and visa versa and hence neither will get freed.

```
Destructor TDGHNotificationsIDENotifier.Destroy;

Var
  iModule : Integer;

Begin
  For iModule := FModuleNotifierRefs.Count - 1 DownTo 0 Do
    Begin
      {$IFDEF DEBUG}
      CodeSite.Send('Destroy', FModuleNotifierRefs[iModule].FileName);
      {$ENDIF}
      FModuleNotifierRefs.Delete(iModule);
      //: @note Cannot remove any left over notifiers here as the module
      //:       is most likely closed at this point.
    End;
  FModuleNotifierRefs.Free;
  Inherited Destroy;
End;
```

Now for the final part. In the `FileNotification` method I've added some code to look for `ofnFileOpened` and `ofnFileClosing` notitications. In the `ofFileOpened` notification I check whether the file module implements the `IOTAProject` interface as this determines if its a Project or Project Group and if so I add and Project Notifier else I'll create a Module Notifier and add then to the collection.

In the `ofnFileClosing` notification I look to the index associated with the file name and remove it from the IDE accordingly.

```
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;
  iModuleIndex: Integer;
  P : IOTAProject;
  eNotiferType : TDGHIDENotification;
  R: TModNotRec;
  MN : TDNModuleNotifier;

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);
              iModuleIndex := M.AddNotifier(MN);
              eNotiferType := dinProjectNotifier;
            End Else
            Begin
              MN := TDNModuleNotifier.Create('IOTAModuleNotifier', FileName, dinModuleNotifier);
              iModuleIndex := M.AddNotifier(MN);
              eNotiferType := dinModuleNotifier;
            End;
          FModuleNotifierRefs.Add(TModNotRec.Create(FileName, iModuleIndex, eNotiferType));
        End;
      ofnFileClosing:
        Begin
          M := MS.OpenModule(FileName);
          If Find(M.FileName, iModuleIndex) Then
            Begin
              R := FModuleNotifierRefs[iModuleIndex];
              M.RemoveNotifier(R.NotifierIndex);
              FModuleNotifierRefs.Delete(iModuleIndex);
            End;
        End;
    End;
End;
```

The Find method is as follows:

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

Var
  iModNotIdx : Integer;
  R: TModNotRec;

Begin
  Result := False;
  iIndex := -1;
  For iModNotIdx := 0 To FModuleNotifierRefs.Count - 1 Do
    Begin
      R := FModuleNotifierRefs.Items[iModNotIdx];
      If CompareText(R.FileName, strFileName) = 0 Then
        Begin
          iIndex := iModNotIdx;
          Result := True;
          Break;
        End;
    End;
End;
```

I hope all of the above is straight forward. The code and the binaries can be found with on the IDE Notifications page.

regards
Dave

Related posts:

1. Notify Me of Everything… – Part 2.1 (Rename Fix) (18.3)
2. Notify me of everything… – Part 1 (10.8)
3. Chapter 7.1: IDE Compilation Events – Revisited… (8.6)
4. Chapter 7: IDE Compilation Events (7.2)
5. Chapter 5: Useful Open Tools Utility Functions (6.7)

Category: IDE Notifications Open Tools API RAD Studio