Dave's Development Blog

Software Development using Borland / Codegear / Embarcadero RAD Studio

embarcadero MVP RAD Studio

embarcadero BUILT FOR RX RAD Studio

How to get the current compiling settings

By David | December 1, 2019 0 Comment

In most instances checking that an `IOTAProject` interface supports `IOTAProjectBuildConfigurations` and using the interface's `ActiveConfiguration` to get the configuration being used works as the IDE changes the active project. However there is one circumstance where this does not work and that is with project Build Groups which are accessed from the Project Manager and allow you to build selected projects in the group with specific configurations.

So after some investigation I found that the only reliable way to get the platform and configuration is to implement an `IOTAProjectCompileNotifier` interface and add it to the project. Below is the definition of the class which implements the notifier (this code can be found within the my Integrated Testing Helper on GitHub under the Development branch).

```
Type
  TITHProjectCompileNotifier = Class(TNotifierObject, IOTAProjectCompileNotifier)
  Strict Private
    FCompileInformation : IITHCompileInformation;
  Strict Protected
    Procedure AfterCompile(Var CompileInfo: TOTAProjectCompileInfo);
    Procedure BeforeCompile(Var CompileInfo: TOTAProjectCompileInfo);
  Public
    Constructor Create(Const CompileInformation : IITHCompileInformation);
    Destructor Destroy; Override;
  End;
```

You will notice that besides implementing the methods of the interface the constructor passes an interface you will not recognise and there is also a field of the same type. This interface is defined below and allows the notifier to return information back to the application when the notifier methods are invoked by the IDE.

```
Type
  IITHCompileInformation = Interface
    ['{16E8A111-2F16-4516-B526-85EE284FBB03}']
    Function  GetCompileInformation : TOTAProjectCompileInfo;
    Procedure SetCompileInformation(Const CompileInfo : TOTAProjectCompileInfo);
    Property CompileInformation : TOTAProjectCompileInfo Read GetCompileInformation
Write SetCompileInformation;
  End;
```

This interface provides a simple property for reading and writing a `TOTAProjectCompileInfo` record. If you've not used this record before, it is defined in the `ToolsAPI.pas` file as follow:

```
Type
  TOTAProjectCompileInfo = record
    Mode: TOTACompileMode;
    Configuration: string;
    Platform: string;
    Result: Boolean;
  end;
```

As you can see it provides all the information you need to know about the compiling action about to start or that has just finished when returned from the above `IOTAProjectCompileNotifier` interface. So the following is the implementation of the `IOTAProjectCompileNotifier`:

```
Procedure TITHProjectCompileNotifier.AfterCompile(Var CompileInfo:
TOTAProjectCompileInfo);

Begin
End;

Procedure TITHProjectCompileNotifier.BeforeCompile(Var CompileInfo:
TOTAProjectCompileInfo);
Begin
  FCompileInformation.CompileInformation := CompileInfo;
End;

Constructor TITHProjectCompileNotifier.Create(Const CompileInformation :
IITHCompileInformation);

Begin
  {$IFDEF CODESITE}CodeSite.TraceMethod(Self, 'Create', tmoTiming);{$ENDIF}
  FCompileInformation := CompileInformation;
End;

Destructor TITHProjectCompileNotifier.Destroy;

Begin
  {$IFDEF CODESITE}CodeSite.TraceMethod(Self, 'Destroy', tmoTiming);{$ENDIF}
  Inherited Destroy;
```

```
End;
```

The constructor simply saves the passed interface for later use in the `BeforeCompile` method. The destructor here is not necessarily needed how ever I usually create one (along with a constructor) and place CodeSite tracing in them so that I can check that the class is being constructed and more importantly destroyed at the right times. You will notice that I only have code in the `BeforeCompile` method as this it the best place for me to get this information as returning it in `AfterCompile` provide no benefit.

So that provides us with the information but where do we create these notifiers? Well in the same way we created `IOTAProjectNotifier`'s in the article Notify Me of Everything – Part 2.

So as before we use the `FileNotification` method of the `IOTAIDENotifier` interface as below.

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

Var
  MS : IOTAModuleServices;
  M: IOTAModule;

Begin
  If Not Cancel And Supports(BorlandIDEServices, IOTAModuleServices, MS) Then
    Case NotifyCode Of
      ofnFileOpened:
        Begin
          M := MS.FindModule(FileName);
          InstallNotifier(M);
        End;
      ofnFileClosing:
        Begin
          M := MS.FindModule(FileName);
          UninstallNotifiers(M);
        End;
    End;
End;
```

So when I implemented the above code I decided to clean it up with two methods that do the work as follows:

```
Procedure TITHelperIDENotifier.InstallNotifier(Const M: IOTAModule);

Var
  P: IOTAProject;
  PN: IOTAProjectNotifier;
  iModuleIndex: Integer;
  {$IFDEF DXE00}
  PCN: TITHProjectCompileNotifier;
  {$ENDIF DXE00}

Begin
  If Supports(M, IOTAProject, P) Then
    Begin
      ...
      {$IFDEF DXE00}
      If Assigned(P.ProjectBuilder) Then
        Begin
          PCN := TITHProjectCompileNotifier.Create(FCompileInformation);
          iModuleIndex := P.ProjectBuilder.AddCompileNotifier(PCN);
          FProjectCompileNotifierList.Add(M.FileName, iModuleIndex);
        End;
      {$ENDIF DXE00}
    End;
End;
```

The above code queries the module interface for a `IOTAProject` interface and if it exists we check that the project has a `ProjectBuilder` (`IOTAProjectBuilder`) interface as this contains the methods to adding and removing the `IOTAProjectCompileNotifier`. With this interface we create and install our notifier into the IDE and save the returned index in a collection so we can later retrieve the index associated with the name of the project.

The removal of the notifier is similar as below:

```
Procedure TITHelperIDENotifier.UninstallNotifiers(Const M: IOTAModule);

Var
  P: IOTAProject;
  iModuleIndex: Integer;

Begin
  If Supports(M, IOTAProject, P) Then
    Begin
      ...
      {$IFDEF DXE00}
      If Assigned(P.ProjectBuilder) Then
        Begin
          iModuleIndex := FProjectCompileNotifierList.Remove(M.FileName);
          P.ProjectBuilder.RemoveCompileNotifier(iModuleIndex);
        End;
      {$ENDIF DXE00}
    End;
End;
```

The code contains `IFDEF`s as this interface (`IOTAProjectCompileNotifier`) is not available in RAD

Studio 2010 and below.

One thing this code does not do at the moment is handle the renaming of a project (hence in the Development branch). The under-pinnings are there in the notifier lists. If you want to know how to do this then read the article Notify Me of Everything – Part 2.1.

regards
Dave.

Category: Delphi Integrated Testing helper Open Tools API RAD Studio Tags: IOTAIDENotifier, IOTAProject, IOTAProjectBuilder, IOTAProjectCompileNotifier