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

Chapter 12: Project Repository Wizards

By David | September 7, 2011 4 Comments

In this chapter I'm going to describe how to implement a Project Wizard which appears in the File | New or File | New | Other dialogue depending on the version of Delphi you have. I've actually had to work this one out from scratch as the last time I did this was with Delphi 3 which used a different method for creating wizards.

This particular wizard will either appear as an icon in a tab of its own for earlier version of Delphi or under a new branch in modern versions of Delphi. These types of wizard are useful for creating new projects in their entirety.

Below is the code to create the wizard interface and it contains quite a number of interfaces. The first `IOTAWizard` should be obvious from previous example but this is then followed by various `IOTARepositoryWizard` and `IOTAProjectWizard` interfaces. The different version of the interfaces are for different version of the IDE and their methods will be explain later. To implement a Project Wizard you must also implement the Repository Wizard interfaces as well.

```
Type
  TRepositoryWizardInterface = Class(TNotifierObject, IOTAWizard, IOTARepositoryWizard
    {$IFDEF D0006}, IOTARepositoryWizard60 {$ENDIF}
    {$IFDEF D2005}, IOTARepositoryWizard80 {$ENDIF},
    IOTAProjectWizard
    {$IFDEF D2005}, IOTAProjectWizard100 {$ENDIF})
  {$IFDEF D2005} Strict {$ENDIF} Private
  {$IFDEF D2005} Strict {$ENDIF} Protected
  Public
    {$IFDEF D2005}
    Constructor Create;
    {$ENDIF}
    // IOTAWizard
    Procedure Execute;
    Function GetIDString: String;
    Function GetName: String;
    Function GetState: TWizardState;
    Procedure AfterSave;
```

```
    Procedure BeforeSave;
    Procedure Destroyed;
    Procedure Modified;
    // IOTARepositoryWizard
    Function GetAuthor: String;
    Function GetComment: String;
    {$IFDEF D0006}
    Function GetGlyph: Cardinal;
    {$ELSE}
    Function GetGlyph: HICON;
    {$ENDIF}
    Function GetPage: String;
    {$IFDEF D0006}
    // IOTARepositoryWizard60
    Function GetDesigner: String;
    {$ENDIF}
    {$IFDEF D2005}
    // IOTARepositoryWizard80
    Function GetGalleryCategory: IOTAGalleryCategory;
    Function GetPersonality: String;
    {$ENDIF}
    // IOTAProjectWizard
    {$IFDEF D2005}
    // IOTAProjectWizard100
    Function IsVisible(Project: IOTAProject): Boolean;
    {$ENDIF}
  End;
```

There is one thing to not here. The `GetGlyph` declaration has changed since Delphi 5.

Below I'm going to walk through each method of the above definition so you can understand what needs coding and what doesn't and how they should be implemented.

Below are some resource strings. These are not need for the OTA but simply for me to implement some messages in the code to help workout when methods are called.

```
{$IFDEF D0006}
ResourceString
  strRepositoryWizardGroup = 'Repository Wizard Messages';
{$ENDIF}
{$IFDEF D2005}
ResourceString
  strMyCustomCategory = 'OTA Custom Gallery Category';
{$ENDIF}
```

This is a method of the `IOTAWizard` interface and as far as I can tell does not get called for this type of wizard.

```
Procedure TRepositoryWizardInterface.AfterSave;

Begin
  OutputMessage('AfterSave' {$IFDEF D0006}, strRepositoryWizardGroup {$ENDIF});
End;
```

This is a method of the **IOTAWizard** interface and as far as I can tell does not get called for this type of wizard.

```
Procedure TRepositoryWizardInterface.BeforeSave;

Begin
  OutputMessage('BeforeSave' {$IFDEF D0006}, strRepositoryWizardGroup {$ENDIF});
End;
```

This create method is only implemented for Delphi 2005 and above as the IDE works differently from earlier versions. This constructor creates a new Category in the gallery under which the project Wizard is installed. For Delphi 2005 and above this is the method that should be used not the below **GetPage** method from older version of the IDE. It simply adds a new category to the IDE with the Delphi New Category as its parent.

```
{$IFDEF D2005}
Constructor TRepositoryWizardInterface.Create;

Begin
  With (BorlandIDEServices As IOTAGalleryCategoryManager) Do
    Begin
      AddCategory(FindCategory(sCategoryDelphiNew), strMyCustomCategory,
        'OTA Custom Gallery Category');
    End;
End;
{$ENDIF}
```

This is a method of the **IOTAWizard** interface and as far as I can tell does not get called for this type of wizard.

```
Procedure TRepositoryWizardInterface.Destroyed;

Begin
  OutputMessage('Destroyed' {$IFDEF D0006}, strRepositoryWizardGroup {$ENDIF});
End;
```

This method is executed when the Project Wizard is selected from the Gallery and this is where we will in future chapters implement the creation of a project.

```
Procedure TRepositoryWizardInterface.Execute;

Begin
  ShowMessage('Hello OTA Example from the Project Repository Wizard.');
End;
```

This is a method of the **IOTAWizard** interface and returns the Author of the wizard.

```
Function TRepositoryWizardInterface.GetAuthor: String;

Begin
  Result := 'David Hoyle';
End;
```

This is a method of the **IOTAWizard** interface and returns a comment for the wizard.

```
Function TRepositoryWizardInterface.GetComment: String;

Begin
  Result := 'This is an example of an OTA Repository Wizard';
End;
```

This is a method of the **IOTARepositoryWizard60** interface and returns the type of designer to be used. This is due to the IDEs of the time being able to target Linux. For this we just returns the constant string for the VCL. This will perhaps change in Delphi XE2 and the targeting of the Mac OS.

```
{$IFDEF D0006}
Function TRepositoryWizardInterface.GetDesigner: String;

Begin
  Result := dVCL;
End;
{$ENDIF}
```

This is a method of the **IOTARepositoryWizard80** interface and specifies under which category in the gallery this new Project Wizard will appear. In this case it appears under the new category we created in the constructor of our wizard.

```
{$IFDEF D2005}
Function TRepositoryWizardInterface.GetGalleryCategory: IOTAGalleryCategory;

Begin
  Result := (BorlandIDEServices As
IOTAGalleryCategoryManager).FindCategory(strMyCustomCategory);
End;
{$ENDIF}
```

This is a method of the **IOTARepositoryWizard** interface and defines the ICON handle to be used for the Project Wizard. In testing I've ascertained that this can ONLY be an ICON and not a bitmap. I should have guessed from the original signature. All we do here is return the handle of an ICON in a resource bound to the Package or DLL. You can see how this is done by looking at the source code at the end of this article.

```
{$IFDEF D0006}
Function TRepositoryWizardInterface.GetGlyph: Cardinal;
{$ELSE}
Function TRepositoryWizardInterface.GetGlyph: HICON;
{$ENDIF}

Begin
  Result := LoadIcon(hInstance, 'RepositoryWizardProjectIcon')
End;
```

This is a method of the **IOTAWizard** interface and returns the ID string of the wizard. This must be unique especially in a project that contains multiple wizards.

```
Function TRepositoryWizardInterface.GetIDString: String;

Begin
  Result := 'OTA.Repository.Wizard.Example';
End;
```

This is a method of the **IOTAWizard** interface and returns the name of the wizard.

```
Function TRepositoryWizardInterface.GetName: String;

Begin
  Result := 'OTA Repository Wizard Example';
End;
```

This is an method of the **IOTARepositoryWizard** interface and is required for earlier version of Delphi (before 2005) in order to tell the IDE on which page (tab) the Project Wizard should appear.

```
Function TRepositoryWizardInterface.GetPage: String;

Begin
  Result := 'OTA Examples';
End;
```

This is a method of the **IOTARepositoryWizard80** interface and tells the IDE which personality the Project belongs to (Delphi, C++ Builder, etc).

```
{$IFDEF D2005}
Function TRepositoryWizardInterface.GetPersonality: String;

Begin
  Result := sDelphiPersonality;
End;
{$ENDIF}
```

This is an method of the **IOTAWizard** interface and returns that the wizard is enabled.

```
Function TRepositoryWizardInterface.GetState: TWizardState;

Begin
  Result := [wsEnabled];
End;
```

This is a method of the **IOTAProjectWizard100** interface which signifies that the wizard is visible for all projects. You may wish to disable a project wizard for a particular given project.

```
{$IFDEF D2005}
Function TRepositoryWizardInterface.IsVisible(Project: IOTAProject): Boolean;

Begin
  Result := True;
End;
{$ENDIF}
```

This is an method of the **IOTAWizard** interface and as far as I can tell does not get called for this type of

wizard.

```
Procedure TRepositoryWizardInterface.Modified;

Begin
  OutputMessage('Modified' {$IFDEF D0006}, strRepositoryWizardGroup {$ENDIF});
End;
```

Finally we need to remove any message from the IDE before we unload. This is only because I'm output messages with the library routines. If you don't use message that implement **IOTACustomMessages** you do not require this but I always think its a safe thing to do.

```
Initialization
Finalization
  ClearMessages([cmCompiler..cmTool]);
End.
```

Obviously we need to create our wizard so the following code is added to the **InitialiseWizard** function. Note that this isn't the main wizard for this example project and therefore is not passed back to the DLL loading code. If it were the main wizard then you would only use **AddWizard** in a Package and **RegisterProc** called from **InitWizard** in a DLL. See the post Fatal Mistake in DLL… Doh! for more details

```
Function InitialiseWizard(WizardType : TWizardType) : TWizardTemplate;

Var
  Svcs : IOTAServices;

Begin
  ...
  // Create Project Repository Interface
  iRepositoryWizardIndex := (BorlandIDEServices As IOTAWizardServices).AddWizard(
    TRepositoryWizardInterface.Create);
End;
```

And of course we need to unload the wizard on unloading the package.

```
  ...
Initialization
  ...
Finalization
```

```
    ...
    // Remove Repository Wizard Interface
    If iRepositoryWizardIndex <> 0 Then
      (BorlandIDEServices As IOTAWizardServices).RemoveWizard(iRepositoryWizardIndex);
  End.
```

Hope this provides helpful. We will use this building block for the next chapter where we'll create a project's code.

This code for this project is here (OTA Chapter 12).

Dave.

Category: Open Tools API Tags: Borland, BorlandIDEServices, CodeGear, Delphi, Embarcadero, Experts, IOTAProjectWizard, IOTAProjectWizard100, IOTARepositoryWizard, IOTARepositoryWizard60, IOTARepositoryWizard80, IOTAWizard, IOTAWizardServices, OTA, RAD Studio

4 thoughts on "Chapter 12: Project Repository Wizards"

Kevin G. McCoy
October 17, 2011

Very informative, and thanks for this excellent reference! Could you please expand on the TRepositoryWizardInterface.Execute method with emphasis on how one goes about inserting more than one unit into a project?

I have seen lots of examples for how one inserts a single unit into a project using OTA, but not how you do more than one in a single Wizard operation.

I need to know how to insert the source and DFM info of several actual units/forms into a project.

I also need to add a number of supporting units/forms that are used as base classes for the actual units. I need to add these to the project to make my version control system happy and to keep the IDE from gaaking on missing ancestor classes.

Best regards,

Kevin G. McCoy

David Post author

October 17, 2011

Kevin,

Thanks for this. Unfortunately I've not got around to writing the next bit about creating units (including forms) but you can find examples of how to do this in my BrowseAndDocIt (http://www.davidghoyle.co.uk/browseanddocit.html) code (see the DUnitCreator.pas file). As for adding several units/forms at a time its simply repetitive calls to ancestors of IOTAModuleCreator where each creates a different unit. If you parameterise the class you can use the same class for different units and forms. I'll endeavour to write the next bit very soon.

Dave.

Kevin G. McCoy
October 18, 2011

Dave,

Thanks for getting back to me.

After posting, I figured out the IOTAModuleCreator trick. I am a bit concerned about freeing memory after creating a bunch of units/forms. Does IOTAModuleCreator free any resources you pass it, or am I responsible for that? I don't want to end up with a whopping big memory leak. I also don't want to double-free, if OTA is freeing resources behind the scenes.

I will take a look at your example code and try to glean some ideas.

Best regards,

Kevin G. McCoy

David Post author
October 19, 2011

Sometimes I have to remind myself by looking at my own code 🙂

This depends upon what exactly you are referring to. If you are referring to the class which implements IOTAModuleCreator then you do no need to FREE it, just make all referenced to it NIL so that the reference count drops to zero and the IDE will disposed of the memory. If you are referring to object passed to the class which implements the creator, then these will need to the freed manually unless they also implement an IOTA interface in which case they are reference counted and once the count drops to zero they will also be disposed of. If its something else, let me know the specifics and I'll try and help more.

regards
Dave.

Comments are closed.