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

RegisterPackageWizard

By David | April 17, 2016 0 Commen

Confession

I have to confess that I don't know why I've never used `RegisterPackageWizard` when creating package wizards. It seems to have been available for ever (I can only check back to D5) and looks to be the intended way that package wizards should be created.
I suspect that apart from my IDE Explorer and AutoSave experts (which have a very old heritage) I've never needed to build package wizards thus this is probably why I've never looked into this. Also why would the `IOTAWizardServices` have methods `AddWizard` and `RemoveWizard`?

Behaviour

To satisfy myself that the call to `RegisterPackageWizard` manages the entire life time of a wizard I created a very simple package wizard and found that the RAD Studio XE10 IDE creates the wizard in the Register procedure when a call is made to the `RegisterPackageWizard` method and when the package is unloaded, i.e. by the IDE closing or the package being uninstalled / removed, the wizard is destroyed without any additional code.

```
Procedure Register;

Begin
  RegisterPackageWizard(TMyTestWizard.Create);
End;
```

I also repeated the exercise with a DLL and `InitWizard` and again the IDE managed the entire life time of the wizard. This was interesting as although I coded up the removal of wizard interfaces for my DLLs I had it in the back of my mind that they were not unloaded. Don't know why I thought this but I did.

So am I going to change all my current code? No. Since it works as is and I'm fairly sure the IDE just does all the same thing I did manually there's no need to change however I am going to update the IDE Explorer wizard using this alternate method while I rebuild it to use the new RTTI module and provide loads more details about the IDE.

How would this change your wizard/expert

In the examples I've provided I've used a single module to contain all the creation and destruction code for wizard, experts, notifiers, etc. This wouldn't necessarily need to be done as your main wizard would behaviour as a collaborative object and maintain all the other OTA instances for you, i.e. all the creation code would go into the constructor for your main wizard and all the destruction code would go into the destructor for your main wizard. The only code that could remain outside would be our class method calls to create the dockable forms.

Why

So why did I stumble on this now and not before? Well I've been looking at doing an example of an OTA wizard/expert entirely coded in C++ Builder and more probably will re-implement the entire OTATemplate example in C++ so that those of you who would like to do OTA in C++ can do so. Starting with C++ Builder 6 there was some information on building experts in the Developer's Guide which has survived until today in the various RAD Studio documentation system. C++ lacks a language equivalent of Initialization and Finalization which is where most of my creation and destruction code current resides hence the need to look at different ways to do things.

No related posts.

Category: Open Tools API