Contents

- Add a Model project
- Delete any existing file links in the client application project
- Finish the Model project
 - Fix Model namespace references
 - How to postpone namespace update
 - Build the Model project
- Add a Client Model project
- Finish the Client Application
- Finish the .NET Client Application

The tutorials instruct you to add your entity model to one of the projects generated from a DevForce template.

They do so because they are tutorials. The entity model tends to be in its own project/assembly in production applications. This topic demonstrates how to **break out the entity model** into its own assembly.

The tutorials encourage you to add your entity model to one of the existing projects: the Web Application project in n-tier examples; the lone application project in the 2-tier samples.

As your application evolves and begins to add other projects that need to reference the entity model, you may wish you had put the entity model in its own project. You don't have to start over; it's pretty easy to extract it as its own project and re-link the projects that depend upon it.

Here you will find step-by-step instructions.

We provide a Silverlight sample application showing the "before" and "after" C# solutions.

Be sure to backup the entire solution before you begin.

Add a Model project

The new entity model project is a full .NET 4.5 class library.

Starting in Visual Studio 2012:

- From the menu: File | Add | New Project...
- Select the Class Library template
- Ensure the top-left ComboBox says ".NET Framework 4.5"
- Name it to match the entity model's namespace; we'll call ours "DomainModel"
- Delete Class I

Move the Entity Framework model (*.edmx*), the *.tt* file, and all custom model classes from the current project (either the application project in a 2-tier solution or the web project in an n-tier solution) into the new *DomainModel* project.

- Select each model-file (you can do all at once with ctrl-mouse-click)
- Drag them to DomainModel
- Delete these files from the original project (reminder: back-up the solution first)
- Add a *DomainModel* project reference to the original application project! This step is easy to forget, but you want to be sure to reference your new assembly.

Delete any existing file links in the client application project

Existing links to your entity model are now broken and likely display the yellow warning triangle icon.

· Remove all links to the entity model class files

You must delete those links before proceeding.

Finish the Model project

You should have a .tt file and a DevForce code-generated file under that .tt file.

You probably **also** have a code-generated file appears **under** the *.edmx* file, that's the Entity Framework getting involved when it should not.

- Open the "Properties" window for the .edmx file by typing Alt-Enter
- Notice that the "Custom Tool" entry says EntityModelCodeGenerator.
 - Clear the "Custom Tool" entry (it should be empty).

- The Entity Framework generated code file disappears
- Right-mouse-click the .*tt* file
- Pick "Run Custom Tool"

This regenerates the code file under the *DomainModel* namespace and adds the minimally required references at the same time:

- IdeaBlade.Core
- IdeaBlade.EntityModel
- IdeaBlade.Validation
- .NET's System. ComponentModel. DataAnnotations

You can use the DevForce 2012 Client NuGet package to add these references.

Make sure you moved over all of the custom model files that you wrote (e.g., custom entity class files and metadata files).

Fix Model namespace references

The re-located, re-generated entity model classes are now defined in the default namespace for the model project, *DomainModel*. That is typically what you want.

It also means that your **model namespace references elsewhere are stranded under the old name**. You have to clean them up.

Start with the custom classes you moved into the *DomainModel* project. A *Customer* partial class, for example, is still defined under its old namespace. Use search-and-replace to swap in the new namespace for all of the custom classes **in this project only**.

Do not search-and-replace the namespace in other projects, especially not your application project. The old namespace name remains legitimate for non-model classes. You are likely to succeed simply by adding "*using DomainModel;*" statements to the class files that need them. Let the compiler be your guide. Be patient for now.

How to postpone namespace update

You can postpone this exercise by resetting the default namespace for the *DomainModel* project to the old namespace name as it was before we moved these files. That namespace is probably the default namespace of the Web Application project.

- Open the properties editor for the DomainModel project
- On the "Application" tab, set the "Default namespace" to the previous model namespace value
- Close that tab
- Right-mouse-click the .tt file
- Pick "Run Custom Tool"

The entity class file is re-generated again, restoring the entity classes to their old namespace.

You can come back and fix this later. The longer you wait, the more you'll have to clean up.

Build the Model project

Build the *DomainModel* project. It should build cleanly or be easy to fix as it has no dependencies on other projects in your solution.

If this is a .NET client application, you're almost done. Skip ahead to the "Finish the .NET Client Application" section.

Add a Client Model project

This section applies only to Silverlight and Windows Store applications. .NET client applications can simply reference the *DomainModel* project and do not usually require a separate client assembly.

In this topic, our assumption is that you want a separate client application project to parallel the separate full .NET *DomainModel* project we just created.

- From the menu: File | Add | New Project...
- · Select the Silverlight Class Library or Windows Store Class Library template
- Name it to match the entity model's name with a .SL or .WinRT suffix; we'll call our Silverlight library "DomainModel.SL"
- Delete Class1
- Press Alt-Enter on the new project to open its Properties
- Set the "Default namespace:" to *DomainModel* (i.e., remove the suffix.

Now you're ready to link to the entity model class files in the DomainModel project.

- · Press Alt-Shift-A to launch the "Add Existing Item" dialog
- Navigate to the DomainModel project that holds the full .NET model files
- Select every file **code** file (.cs or .vb); **do not** select any other file type.
- Do not click the "Add" button
- Drop down the ComboBox next to the "Add" button and pick "Add As Link".

	<u>A</u> dd ▼		Cancel
		Add	
	1	Add As Link	
	(See	Show previous versions	

Add Silverlight assembly references:

- Add DevForce references
 - IdeaBlade.Core
 - IdeaBlade.EntityModel
 - IdeaBlade.Validation
 - IdeaBlade.Linq (if needed)
- Select all of them and open the "Properties Window"
- Set their "Specific Version" to False
- · Add .NET references
 - System.ComponentModel.DataAnnotations
 - System.Runtime.Serialization

Remember that the DevForce 2012 Client NuGet package will add these references for you.

Build the new client project. It should build cleanly or be easy to fix as it has no dependencies on other projects in your solution.

Finish the Client Application

Almost there!

- Add a reference to the new client DomainModel.SL or DomainModel.WinRT project.
- Build the entire solution, correcting as necessary.
- Run.

The most likely compile errors will be missing references to your re-factored model project. The entity classes have a new namespace. Adding some "*using DomainModel*;" statements to the class files that need them should do the trick.

Finish the .NET Client Application

Almost there!

- Add a reference to the .NET DomainModel project.
- Build the entire solution, correcting as necessary.
- Run.

The most likely compile errors will be missing references to your re-factored model project. The entity classes have a new namespace. Adding some "*using DomainModel;*" statements to the class files that need them should do the trick.