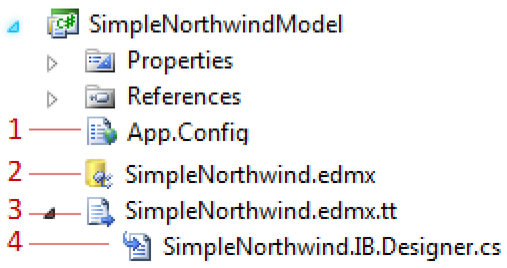
Contents

Generate entity class code from an **Entity Data Model (EDM)** prepared with the Entity Framework EDM Designer.

Alternatively, you could create your entity class model entirely in code using the "Code First" style and bypass the EDM design and generation process described here.

Model project changes

Add an Entity Framework Entity Data Model (EDM) to a project and examine the project in the Visual Studio "Solution Project" window.



The EDM Wizard and DevForce code generation have made significant changes to the project:

1. The EDM Wizard added Entity Framework assembly references; DevForce added its assembly references.
2. The EDM Wizard added an Entity Framework connection string to the *App.config* or to the *Web.config* if had been a web application project.
3. The EDM Wizard added the EDMX file of XML that describes your model. You fleshed out that model with the EDM Designer.
4. DevForce added the code generation template file with the *.tt* extension.
5. DevForce generated the nested source code file with the "*.Designer.cs*".

Remember that you must set the *DevForce Enabled* flag to *True* in the Model Properties window in order for DevForce to generate code. This flag is not enabled by default.

DevForce used the T4 code generator built into Visual Studio to create the source code file.

Source code files are **regenerated automatically every time you save a change** to the EDMX file.

You can also **regenerate manually** by **selecting the *.tt* template file** and choosing **Run Custom Tool** from the context menu.

DevForce code generation template

The T4 code generator is directed by the DevForce code generation template which reads the EDMX and emits entity model source code file(s). The template file name shares the same root name as the EDMX file, *SimpleNorthwind* in this example.

Entity Framework (EF) has its own T4 template (several templates actually). The DevForce template replaces EF's template when DevForce tooling is installed and enabled. You can restore EF code generation, perhaps to see how the code files differ, but your application probably won't work until you re-enable DevForce code generation.

The DevForce template only needs the CSDL section of the EDMX file because the .NET entity classes are entirely described by the conceptual entity model. That's why a DevForce entity class model can be designed in the Model First style which only has a CSDL.

Generated source code files

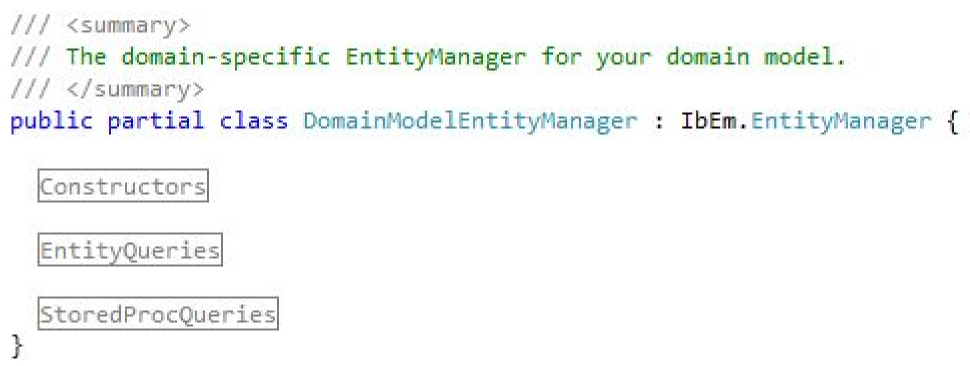
There is only one generated source code file in this small example, *SimpleNorthwind.IB.Designer.cs*. There could be several generated source code files if the model were unusually large. You control the number of classes generated per file - and hence the number of files - by setting the "Max. Classes per File" model property in the EDM Designer.

Inside a source code file

Never modify a generated source code file by hand. When the file is regenerated – as it inevitably will be – all of your changes will be lost. There are other ways to customize the generated source code.

The following snapshots of a generated class file shows three kinds of classes.

At the top is a model-specific EntityManager component. The *EntityManager* is the most important component in all of DevForce and you will get to know it well.

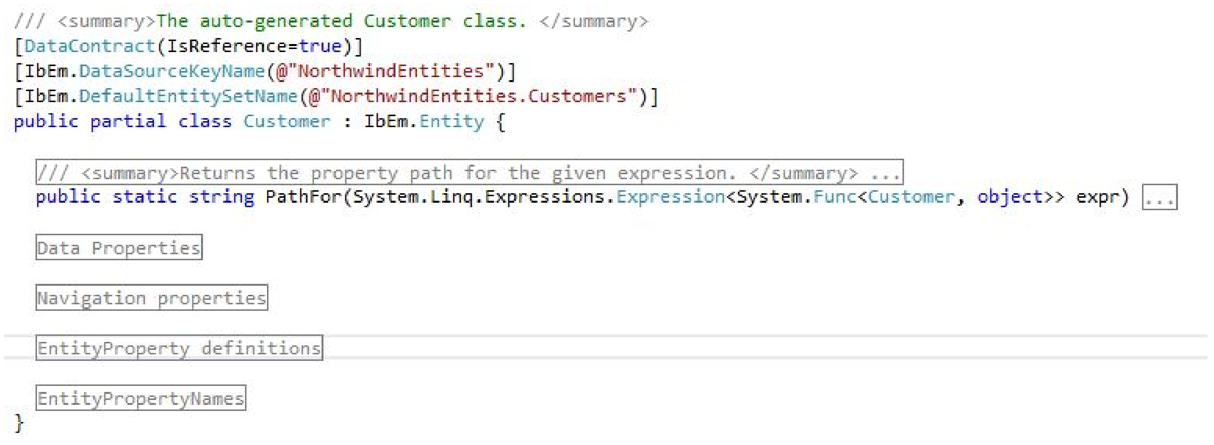


```
/// <summary>
/// The domain-specific EntityManager for your domain model.
/// </summary>
public partial class DomainModelEntityManager : IbEm.EntityManager {

  Constructors

  EntityQueries

  StoredProcQueries
}
```

Below that are the generated entity classes, one after another, each one generated from the conceptual entity defined in your EDM.

DevForce generates true Entity Framework classes. You can perform pure Entity Framework operations with these class using EF's *ObjectContext* in textbook fashion. But the DevForce entity classes differ significantly in their support for distributed application scenarios, validation, property interception, and UI data binding.

The *Customer* class is typical:



```
/// <summary>The auto-generated Customer class. </summary>
[DataContract(IsReference=true)]
[IbEm.DataSourceKeyName(@"NorthwindEntities")]
[IbEm.DefaultEntitySetName(@"NorthwindEntities.Customers")]
public partial class Customer : IbEm.Entity {

  /// <summary>Returns the property path for the given expression. </summary> ...
  public static string PathFor(System.Linq.Expressions.Expression<System.Func<Customer, object>> expr) ...

  Data Properties

  Navigation properties

  EntityProperty definitions

  EntityPropertyNames
}
```

Finally, at the bottom, is the *EntityRelations* class

```
/// <summary>
/// A generated class that returns the relations between entities in this model.
/// </summary>
public partial class EntityRelations : IbEm.IEntityRelations { ...
```

```
''' <summary>
''' A generated class that returns the relations between entities in this model.
''' </summary>
Partial Public Class EntityRelations Inherits IbEm.IEntityRelations ...
```

Within the class are static *EntityRelation* fields defining every relationship between every entity in the model.

Customize the code generation template

In general you customize the generated classes by adding more code to the model. You can trim back some of the emitted code by adjusting the generator control properties within the EDM Designer. You can replace all of the attributes with an entity metadata class.

If these paths prove insufficient, you can customize the code generation template yourself