Contents

Customize the **DevForce code generation template** when you need fine control over the entity class code generated for your Entity Data Model (EDM).

DevForce code generation uses the .NET *Text Template Transformation Toolkit* (T4) to customize code generation. DevForce ships with a default T4 template that is used for all code generation. Like most T4 templates provided by Microsoft and other vendors, this template can be replaced with a custom version. Custom versions are usually a copy of the default template with some minor modifications.

While DevForce supports this approach, we provide what we believe to be a better approach. Instead of modifying the template, you subclass the core template generator class, ***DomainModelTemplate***, overriding only those portions which requires custom treatment. Virtually every major code generation region within your generated code corresponds to a virtual method within this class.

Deriving from *DomainModelTemplate* is much easier than writing raw T4 source code:

- You write in a familiar programming language
- You'll have substantially less code to maintain.
- We may release improved versions of the *DomainModelTemplate* with new features and fixes.
- *DomainModelTemplate* can generate either C# or VB code from a single template file.
- Debugging a DevForce template is substantially easier than debugging a standard T4 template.

The source code for *DomainModelTemplate* is currently not provided with the DevForce installation but is available upon request.

The DevForce template

When you enable DevForce in your Entity Data Model, DevForce adds its T4 template to the project, modifying it slightly to reflect the specifics of your model. A typical example looks something like the following:

```
<#@ template  language="C#" debug="true" hostSpecific="true" #>
<#@ output extension="./DontReadMe" #>
<#@ Assembly Name="IdeaBlade.VisualStudio.OM.CodeGenerator.<efver>.dll" #>
<#@ import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator" #>
<#
 // Source for this file located at:
 // C:\Users\...\AppData\Local\Microsoft\VisualStudio\...\DomainModelTemplate.tt
 // SimpleNorthwind.edmx  <--- Needed so "Transform Related Text Templates On Save" works correctly.
 var template = new DomainModelTemplate(this);
 template.Generate();
#>
<#+
#>
```

Note: *efver* will be either "EF5" or "EF6" depending on the version of Entity Framework you are using. So the resulting assembly reference will be either:
 Assembly Name="IdeaBlade.VisualStudio.OM.CodeGenerator.EF5.dll"
or
 Assembly Name="IdeaBlade.VisualStudio.OM.CodeGenerator.EF6.dll"

The template delegates the work to the *IdeaBlade.VisualStudio.OM.CodeGenerator.**DomainModelTemplate*** class (shown above about five lines from the bottom).

There are three steps to customizing this template:

1. Subclass *DomainModelTemplate* within the existing *.tt* file.
2. Make a template from the modified *.tt* file
3. Move your custom *DomainModelTemplate* to its own assembly.

You get a working template at every step. You can stop after steps #1 or step #2 if you wish. We recommend you go all the way to step #3.

Before starting, you must install the **Visual Studio SDK** to get the *Microsoft.VisualStudio.TextTemplating* library appropriate for your version of Visual Studio.

#1 Subclass inside the *.tt* file

You are going to subclass the *DomainModelTemplate* class inside the project's *.tt* file.

First, add two assembly references:

- Assembly Name="Microsoft.VisualStudio.TextTemplating.<vsver>.0"
- Assembly Name="IdeaBlade.Core.dll"

... and an additional namespace import:

- import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator.Metadata"

Note: *vsver* will depend on your Visual Studio version. For VS 2012 this is "11", while for VS 2013 this is "12".

Next, find the declaration of the *template* variable and set it with your new template generator class

Finally, write the code for your template generator class directly below the old code as seen in this example (using EF 6 and VS 2013):

```
<#@ template  language="C#" debug="true" hostSpecific="true" #>
<#@ output extension="./DontReadMe" #>
<#@ Assembly Name="IdeaBlade.VisualStudio.OM.CodeGenerator.EF6.dll" #>
<#@ Assembly Name="Microsoft.VisualStudio.TextTemplating.12.0" #>
<#@ Assembly Name="IdeaBlade.Core.dll" #>
<#@ import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator" #>
<#@ import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator.Metadata" #>
 ...
 // SimpleNorthwind.edmx  <--- Needed so "Transform Related Text Templates On Save" works correctly.
 var template = new MyTemplate(this);
 template.Generate();
#>
<#+
public class MyTemplate : DomainModelTemplate {
    public MyTemplate(Microsoft.VisualStudio.TextTemplating.TextTransformation textTransformation)
        : base(textTransformation) { }
    protected override void WriteEntityDataPropertyAttributes(EdmPropertyWrapper property) {
        WriteAttribute("Foo(" + Quote(property.Name) + ")");
        base.WriteEntityDataPropertyAttributes(property);
    }
}
#>
```

This customization simply adds a *Foo* attribute to the top of every data entity property. *MyTemplate* replaces *DomainModelTemplate* and overrides the base *WriteEntityDataPropertyAttributes* method to add *Foo*.

If you now right click on the .tt file and select "Run Custom Tool" the model should be regenerated with your customizations.

See the topic on *adding a custom base class* for a more realistic example of a template method override that also makes use of the ***Tag*** *EDM extension property*.

While writing the *MyTemplate* class within the template itself is certainly easy to do, we don't recommended it.

First, this customization only works for the specific EDMX to which it is attached. *SimpleNorthwind.edmx*, the name of this project's EDMX file, is baked into the *.tt* file. If you change the name of the EDMX file you'll have to change this *.tt* file as well. You won't be able to re-use this custom *MyTemplate* class for other models and applications.

Second, you don't get any help from Intellisense while you're writing *MyTemplate*; it's difficult to edit a template file with any significant amount of code this way.

Let's fix the first problem first.

#2 Make a template from the modified *.tt* file

The DevForce EDM Designer Extension looks for a file named *DomainModelTemplate.tt* in a well-known place. *DomainModelTemplate.tt* is actually a template for making *.tt* templates. It has placeholder tokens for model-specific values. It's the file the DevForce Designer Extension used to create the *SimpleNorthwind.edmx.tt* file in the model project.

The goal is to replace the existing *DomainModelTemplate.tt* file with your version so that the next time DevForce generates a entity classes from a model ... any model ... it creates a *.tt* file based on *your* template file.

DomainModelTemplate.tt is in another directory, the one mentioned in a comment of the *.tt* you edited:

```
// Source for this file located at:
// C:\Users\...\AppData\Local\Microsoft\VisualStudio\...\DomainModelTemplate.tt
```

The actual directory location varies according to machine, operating system, and user name.

- Copy the model project *.tt* file that you just edited into template directory
- Backup the current *DomainModelTemplate.tt* by renaming it (e.g., "DomainModelTemplate.tt.backup")
- Rename your modified *.tt* file to *DomainModelTemplate.tt*

Now edit this new version of *DomainModelTemplate.tt* to put in the placeholder tokens.

Change:

```
// SimpleNorthwind.edmx  <--- Needed so "Transform Related Text Templates On Save" works correctly.
```

To:

```
// Source for this file located at: $templateFullFileName$
// $edmxname$  <--- This is needed so that "Transform Related Text Templates On Save" works correctly.
```

All future *.tt* files created for **any** model project will be based on this new template.

You can also update an older project that was generated with the old template version.

- delete its ".tt" file
- make a fake change to the EDMX file (e.g., move an object on the canvas)
- save

This triggers recreation of the EDMX's *.tt* file and regenerates your model using the new template.

#3 Move your custom generator to its own assembly

This third step builds on the second.

The goal is to be able to edit your custom template generator in Visual Studio so you can take advantage of Intellisense, type-checking, and Visual Studio debugging.

You will extract your custom *MyTemplate* class from the template and put it in its own assembly.

Create a new class library project. Let's call it *MyTemplateAssembly*. Then **create the *MyTemplate* class** from the code you wrote earlier.

Add references to

- IdeaBlade.Core.dll
- IdeaBlade.VisualStudio.OM.CodeGenerator.<efver>.dll
- Microsoft.VisualStudio.TextTemplating<vsver>.0.dll

You can find the IdeaBlade assemblies in the folder to which the DevForce EDM Extension was installed, which is usually under C:\Users\<youraccount>\AppData\Local\Microsoft\VisualStudio\<vsver>.0\Extensions. The Microsoft.VisualStudio.TextTemplating assembly will be in either the GAC or the folder to which the VS SDK was installed.

Here's the source code for our new *MyTemplate* class:

```
using IdeaBlade.VisualStudio.OM.CodeGenerator;
using IdeaBlade.VisualStudio.OM.CodeGenerator.Metadata;
namespace MyTemplateAssembly {
public class MyTemplate : DomainModelTemplate {
  public MyTemplate(Microsoft.VisualStudio.TextTemplating.TextTransformation textTransformation)
     : base(textTransformation) { }
  protected override void WriteEntityDataPropertyAttributes(EdmPropertyWrapper property) {
     WriteAttribute("Foo(" + Quote(property.Name) + ")");
    base.WriteEntityDataPropertyAttributes(property);
    }
  }
}
```

```
Imports IdeaBlade.VisualStudio.OM.CodeGenerator
Imports IdeaBlade.VisualStudio.OM.CodeGenerator.Metadata
Public Class MyTemplate
```

```
  Inherits DomainModelTemplate
  Public Sub New(ByVal textTransformation As  _
    Microsoft.VisualStudio.TextTemplating.TextTransformation)
    MyBase.New(textTransformation)
  End Sub
  Protected Overrides Sub WriteEntityDataPropertyAttributes( _
    ByVal property1 As EdmPropertyWrapper)
    WriteAttribute("Foo(" & Quote(property1.Name) & ")")
    MyBase.WriteEntityDataPropertyAttributes(property1)
  End Sub
End Class
```

Now **make a new version** of *DomainModelTemplate.tt* from the original backup copy and **edit this *DomainModelTemplate.tt*** file to use your custom template generator:

- Add a reference to the custom template generator's assembly.
- Import the namespace where your custom template class resides.
- Replace usage of the DevForce *DomainModelTemplate* class with your class.

The final *DomainModelTemplate.tt* would look something like this (for EF 6 and VS 2013):

```
<#@ template  language="C#" debug="true" hostSpecific="true" #>
<#@ output extension="./DontReadMe" #>
<#@ Assembly Name="{path to MyTemplateAssembly}" #>
<#@ Assembly Name="IdeaBlade.VisualStudio.OM.CodeGenerator.EF6.dll" #>
<#@ Assembly Name="Microsoft.VisualStudio.TextTemplating.12.0" #>
<#@ Assembly Name="IdeaBlade.Core.dll" #>
<#@ import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator" #>
<#@ import namespace="IdeaBlade.VisualStudio.OM.CodeGenerator.Metadata" #>
<#@ import namespace="MyTemplateAssembly" #>
<#
 // Source for this file located at: $templateFullFileName$
 // $edmxname$  <--- Needed so "Transform Related Text Templates On Save" works correctly.
 var template = new MyTemplate(this);
 template.Generate();
#>
<#+
#>
```

Notice that you must spell out the full path to your assembly reference. Example: *<#@ Assembly Name="c:\projects \MyExtension\MyExtension.dll" #>*.

Congratulations! You're done! Future *.tt* files created for a model project will be based on your new template **and** you'll be able to maintain that template efficiently in Visual Studio.