Contents

The DevForce T4 code generator emits a partial classes. You can **add capabilities to the generated entity classes** by writing supplemental code in your own **partial class files**.

You add partial class files to the model project, knowing that the compiler will combine the class definitions it finds in all source code files when it builds the final entity classes.

All partial class source code files must be in the same project as the generated class files.

A Sample

To illustrate this technique of extending the generated entity classes, we'll add a *Customer* partial class that enhances the generated *Customer* class in several ways.

Begin by adding a new source code file to the same model project calling it *Customer.cs* (or *Customer.vb*). To this file you could add a *Customer* class that looks like this one:

```
[DebuggerDisplay("{ToString()}")]
[RequiresAuthentication]
public partial class Customer
{
  public Customer()
   {
      CustomerID = SystemGuid.NewGuid();
   }
   [Display(Name = "Original", AutoGenerateField = true, Order = 3)]
  public bool IsOriginal
   {
     get { return CustomerID_OLD != String.Empty; }
   }
  public bool CanDelete { get { return !IsOriginal; } }
  // Enforce integrity of order before adding
  public void AddOrder(Order order) {/*...*/}
  // Enforce rules for removing an order
  public void RemoveOrder(Order order) {/*...*/}
  // Trim spaces from Company name
  [BeforeSet(EntityPropertyNames.CompanyName)]
  public void TrimNameBeforeSet(
      PropertyInterceptorArgs<Customer, String> args)
   {
     if (args.Value != null) args.Value = args.Value.Trim();
   }
  public override string ToString()
   {
     return String.Format("{0}({1})", CompanyName, CustomerID);
   }
}
```

```
<DebuggerDisplay("{ToString()}")>
<RequiresAuthentication> _
Partial Public Class Customer
  Public Sub New()
    CustomerID = SystemGuid.NewGuid()
  End Sub
    <Display(Name := "Original", AutoGenerateField := True, Order := 3)>
     Public ReadOnly Property IsOriginal() As Boolean
Get
  Return CustomerID_OLD <> String.Empty
End Get
     End Property
  Public ReadOnly Property CanDelete() As Boolean
   Get
     Return Not IsOriginal
   End Get
  End Property
  ' Enforce integrity of order before adding
  Public Sub AddOrder(ByVal order As Order) '...
  End Sub
  ' Enforce rules for removing an order
  Public Sub RemoveOrder(ByVal order As Order) '...
  End Sub
  ' Trim spaces from Company name
  <BeforeSet(EntityPropertyNames.CompanyName)>
  Public Sub TrimNameBeforeSet(ByVal args As PropertyInterceptorArgs(Of Customer, String))
    If args.Value IsNot Nothing Then
      args.Value = args.Value.Trim()
    End If
  End Sub
  Public Overrides Function ToString() As String
    Return String.Format("{0}({1})", CompanyName, CustomerID)
  End Function
End Class
```

The example demonstrates many of the kinds of customizations you might make such as:

- The class-level *RequiresAuthentication* attribute - one of several security attributes; - ensures that only authenticated users can query or save *Customer*s.
- An explicit default constructor initializes the *Guid* EntityKey for new customers.
- A custom, read-only *IsOriginal* property that reports if this Customer is one of the original Northwind customers.
- The *Display* attribute hints to the UI how to label and position the *IsOriginal* property.
- *CanDelete*, *AddOrder* and *RemoveOrder* express workflow and integrity rules.
- An attributed property interceptor trims spaces from ends of input *CompanyName*.
- The *ToString* overrides is especially useful during debugging.
- The class-level *DebuggerDisplay* attribute delivers a friendlier value in the debugger.

Adding constructors

Like any .NET class, the compiler imputes a public default, parameterless constructor unless you write a constructor of your own. DevForce doesn't generate a constructor. But you can write one (as shown in the example) and the partial class is a good place to do it.

You might even write a constructor with parameters in which case you must also add a default, parameterless constructor. Another topic covers writing entity constructors in greater detail. For now we'll just call out the importance of having a public default constructor.

Link to the client application

We can't use the .NET model assembly in client environments such as Silverlight, Windows Store or mobile. These applications must have their own version of the entity classes. To ensure the fidelity of definitions in both client and server environments, DevForce adds links to the source code files in the .NET model project to the client application project.
DevForce takes care of linking to the generated source code files automatically. The developer must link to custom source code files manually.

Limits of the partial class

A partial class can **add new** features but it **cannot change existing** features of a generated class. You can't make a *public* property *private*. You can't change its implementation either. But you can "re-attribute" them in a metadata "buddy" class.