

Contents

- [EDM Property Sets](#)
- [Model EDM Properties](#)
 - [EntityManager subclass](#)
 - [Injected base type](#)
 - [Max. classes per file](#)
- [Entity EDM Properties](#)
 - [Query and save authorization example](#)
- [EDM Properties for Properties](#)
 - [Attribute suppression](#)
 - [Concurrency strategy](#)
 - [Entity property accessibility](#)

The [EDM Designer](#) presents a different properties window for each kind of object in the model. This topic describes many of the properties of each conceptual model object types and how to set them.

The EDM Designer Property window lists EDM control properties for EDM objects such as entities, associations, complex types, store objects, and the model itself. Each kind of object has its own list of properties, appropriate to its type.

The most important - and the only properties you can change - are the **Conceptual Model properties**. The [DevForce EDM Extension](#) supplements this set with its own properties to guide DevForce [generation of entity class code](#).

This topic concentrates almost exclusively on the DevForce designer extension properties. You can learn about the base designer properties from [other sources](#).

EDM Property Sets

Five property sets correspond to the five kinds of Conceptual Model objects:

1. The **Model** itself
2. **Entity**
3. **Property**
4. **Association**
5. **Complex Type**

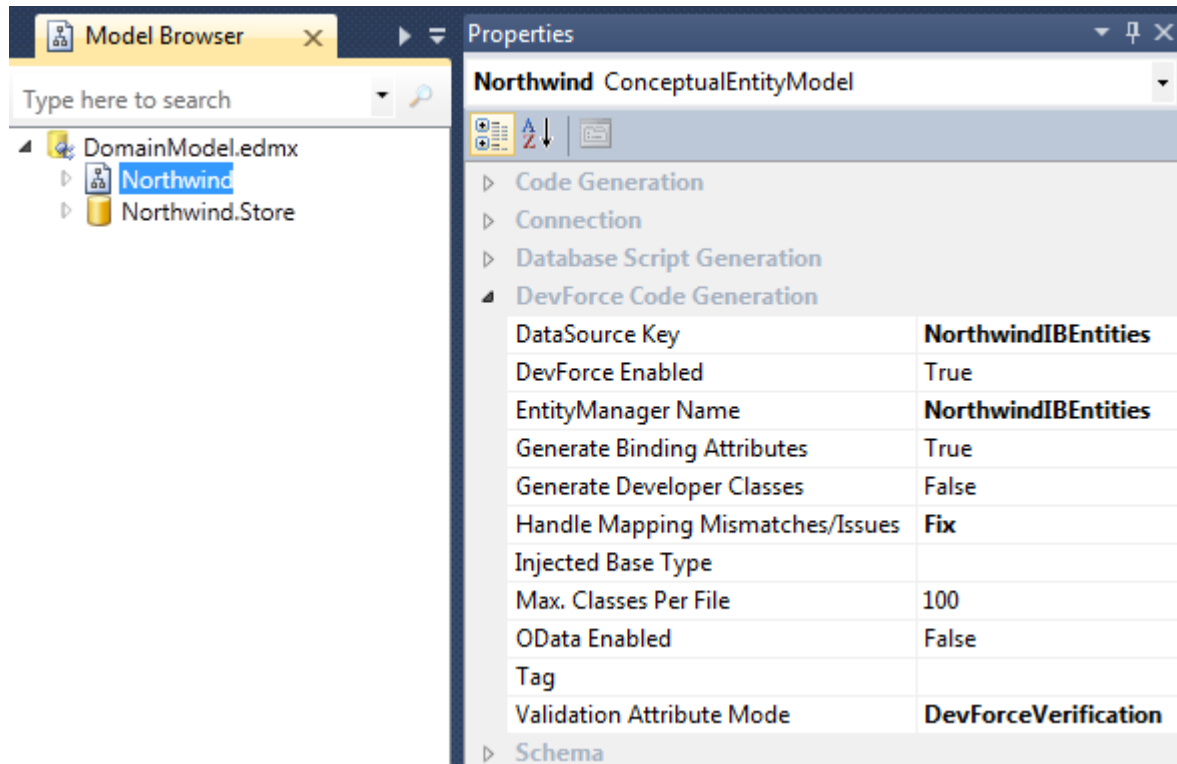
We only consider the first. The Complex Type has a *Tag* property but is otherwise uninteresting. The Association is a worthy topic of its own but has no DevForce-specific EDM properties.

Pin the Properties Window open throughout this exercise and **sort by category**.

Open the [Model Browser](#); it's easier to review a model in the Model Browser than on the [design canvas](#).

Model EDM Properties

Select the Model-level node which is "Northwind" in this screenshot.



Property	Description
DataSource Key	An arbitrary string name that ties this EDM to a single data source. At runtime, DevForce uses this name to find the appropriate database and its connection string. The code generator adorns each entity class with an <i>DataSourceKey</i> attribute that specifies this name, thus binding the entity class to the model's datasource.
DevForce Enabled	Whether DevForce should generate code for this model.
EntityManager Name	Name of the model-specific, <i>EntityManager</i> subclass that is generated for this model; see below .
Generate Binding Attributes	Whether to generate the binding attributes - <i>BindableAttribute</i> , <i>DisplayAttribute</i> , <i>EditableAttribute</i> , and <i>ReadOnlyAttribute</i> - for each entity property. These attributes can be suppressed at the property level .
Generate Developer Classes	Whether to generate for each entity an empty partial class file for developer customizations. <i>False</i> by default. Such files are easy to create manually when needed.
Injected Base Type	Name of the custom base class to inject at the root of the entity model hierarchy; see below .
Max. Classes per File	An integer specifying the maximum number of entity classes to generate in a single class file; see below .
Odata enabled	Whether model-specific <i>EntityManager</i> can be used as an OData DataService. When enabled, DevForce adds attributes and code to the <i>EntityManager</i> to support OData and adds OData library references to the project.
Tag	An arbitrary string of your choosing. Use it to guide your custom code generation .
Validation Attribute Mode	Whether to use DevForce Verification attributes or the .NET attributes from <i>System.ComponentModel.DataAnnotations</i> when adding validation attributes to entity properties. See the "Validation" topic to assess the merits of each choice.

***EntityManager* subclass**

DevForce generates a custom *EntityManager* with model-specific members that derives from the [EntityManager](#). The signature for such an *EntityManager* looks like this:

```
[IbEm.DataSourceKeyName(@"NorthwindIBEntities")]
public partial class NorthwindIBEntities : IbEm.EntityManager { ... }

<IbEm.DataSourceKeyName("NorthwindIBEntities")>
Partial Public Class NorthwindIBEntities Inherits IbEm.EntityManager
End Class
```

The "EntityManager name" matters, especially in a project with multiple EDMX files. If a single project has multiple EDMX files and all of the models share the same "EntityManager Name", then the code generator emits a series of partial class files that compile together to form a single custom *EntityManager* class that spans those models.

If the models each have their own "EntityManager Name", each model gets its own *EntityManager* subclass.

Injected base type

DevForce entity class must ultimately inherit from [Entity](#). If the "Injected Base Type" is blank, the entity classes generated from this model will either inherit directly from *Entity* or from another class in the model.

You can [insert your own base class](#) between *Entity* and the other generated entity classes by naming that class in the "Injected Base Type". There are rules about this class:

- Your base class must inherit from *Entity* or from another class that inherits from *Entity*.
- If the "Injected Base Type" name is a "qualified name" - meaning it contains a period ('.') - then that class is assumed to exist, derives from *Entity*, and is in a referenced assembly of the model project.
- If the "Injected Base Type" is not "qualified" - its name does not contain a period - DevForce generates a partial class file for the base class; the generated class inherits from *Entity* but is otherwise empty.
- In either case, DevForce generates model entity classes that derive from your base class instead of *Entity*.

The reasoning is as follows. Your base class may reside in another project where it can be shared with other models in other projects or even other applications. Such a class would have its own namespace, hence the period in the class name.

On the other hand, you may prefer to define the base class in this project for this model. Generating the base class as a partial class is harmless, guarantees that all model entity classes really do derive from *Entity*, and ensures that the project will compile ... even if you neglect to fill in the base class details in your companion partial class file.

Max. classes per file

Models with a large number of entities can sometimes result in generated code files that are too large to be processed by the Visual Studio editor. This problem may be avoided by generating code into more than one file. The "Max classes per file" setting will limit the number of classes that are generated into a single file and will create as many files as are necessary to meet the specified constraint for each file. The default value for this property is 100 but can be adjusted in either direction. If it is set to 1, then each generated file will contain only a single class.

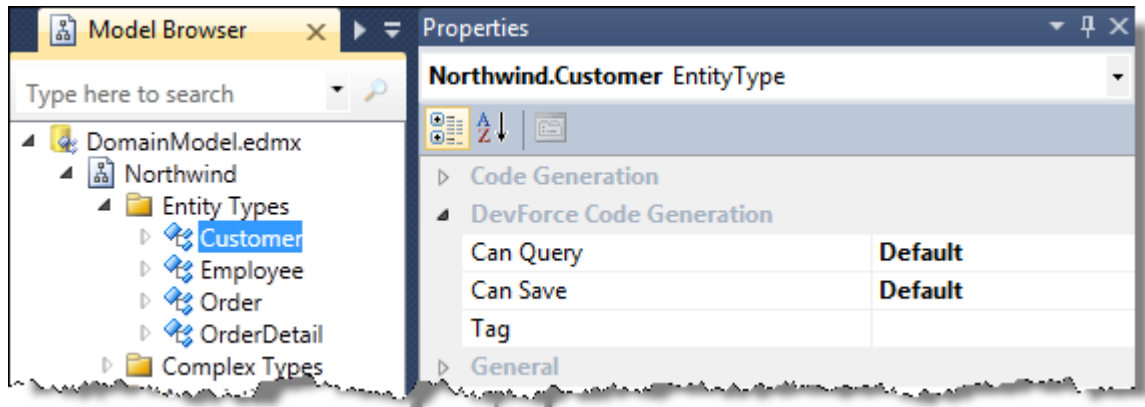
As of 6.1.4, when Max. Classes Per File is set to 1, the generated designer class filename will default to "className.cs" eg: NorthwindIBEntities.cs, EntityRelations.cs, Customer.cs, etc. When **both Max. Classes Per File is set to 1 and Generate Developer Classes is set to True**, the generated designer class filename will default to "className.IB.Designer.cs" eg: NorthwindIBEntities.IB.Designer.cs, EntityRelations.IB.Designer.cs, Customer.IB.Designer.cs. The developer classes name will stay as Customer.cs, Order.cs, etc. Developers have the ability to override the filename suffix by overriding the following virtual property:

```
protected virtual String GeneratedFileNameSuffix {
    get { return "IB.Designer"; }
}
```

Having many generated files is often a bad idea because of issues involved with synchronizing many changed files within a source control system. We recommend leaving this default as is unless you have a specific issue that requires changing it.

Entity EDM Properties

Select an Entity node such as *Customer* as shown in this screenshot.



Property	Description
Can Query	Whether the client is allowed to send a query to the server that returns or involves this type. The choices are <i>True</i> , <i>False</i> , and <i>Default</i> . If <i>True</i> or <i>False</i> , the code generator adds the corresponding ClientCanQueryAttribute to the entity class. <i>Default</i> is equivalent to <i>True</i> and suppresses the attribute altogether. A custom server-side EntityServerQueryInterceptor is the ultimate arbiter of the client's right to query the type.
CanSave	Whether the client is allowed to save entities of this type. The choices are <i>True</i> , <i>False</i> , and <i>Default</i> . If <i>True</i> or <i>False</i> , the code generator adds the corresponding ClientCanSaveAttribute to the entity class. <i>Default</i> is equivalent to <i>True</i> and suppresses the attribute altogether. A custom server-side EntityServerSaveInterceptor is the ultimate arbiter of the client's right to save instances of this type.
Tag	An arbitrary string of your choosing. Use it to guide your custom code generation .

Query and save authorization example

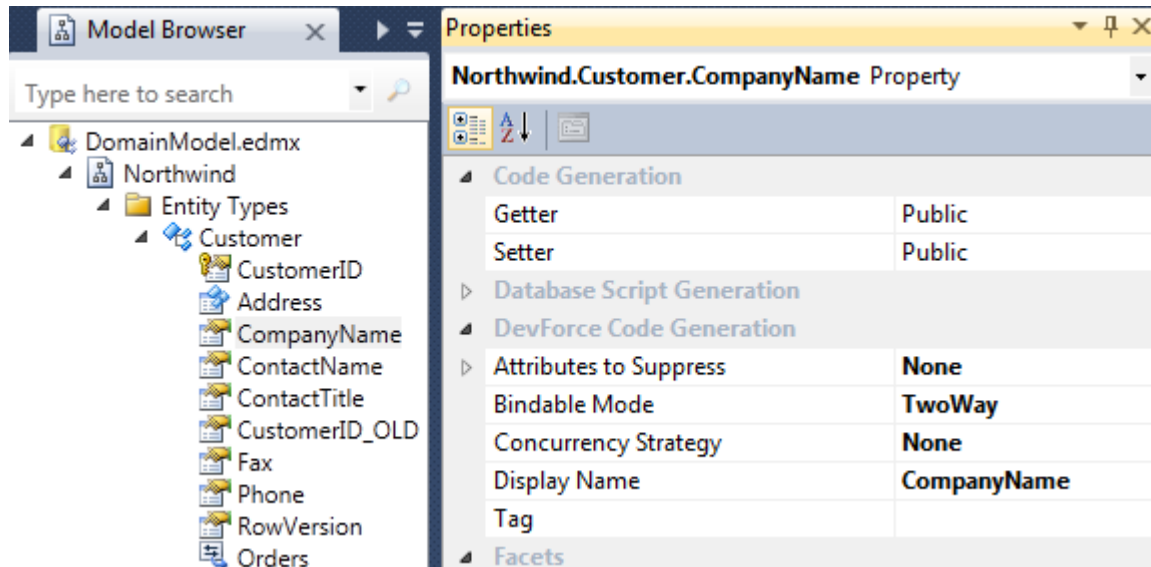
In the following example, the developer specified that the client can query but not save *Customers*. The developer left both authorization properties as *Default* for the *Employee* entity.

```
using IbEm = IdeaBlade.EntityModel;
...
[IbEm.ClientCanQuery(true)]
[IbEm.ClientCanSave(false)]
public partial class Customer : IbEm.Entity { ... }
public partial class Employee: IbEm.Entity { ... }

Imports IbEm = IdeaBlade.EntityModel
...
<IbEm.ClientCanQuery(True)>
<IbEm.ClientCanSave(False)>
Partial Public Class Customer Inherits IbEm.Entity
End Class
Partial Public Class Employee Inherits IbEm.Entity
End Class
```

EDM Properties for Properties

Entity and [ComplexType](#) properties are each defined by a set of EDM properties. The EDM properties for the *Customer.CompanyName* property are below:



Property	Description
Getter	The access mode for the property getter; see note on property accessibility .
Setter	The access mode for the property setter; see note on property accessibility .
Attribute suppression	A collection of settings that determine which of many possible attributes can be generated for this property; see "Attribute suppression" below .
Bindable Mode	A hint to the UI about whether the property can be bound <i>OneWay</i> , <i>TwoWay</i> , or <i>None</i> =not at all. It's the parameter to the <i>DisplayAttribute</i> which is generated for the property unless suppressed .
Concurrency Strategy	Whether this property participates in optimistic concurrency conflict detection and if so how; see below .
Display Name	A hint to the UI about what name to use when constructing a label or grid column header for this property. It's the parameter to the <i>BindableAttribute</i> which is generated for the property unless suppressed .
Tag	An arbitrary string of your choosing. Use it to guide your custom code generation .

Attribute suppression

By default, the code generator adorns entity entity or complex type property with the attributes that are appropriate for that property. The "appropriate" attribute depends upon characteristics of the property such as whether a setter exists, if the property is nullable or not, if it is a string of known length (see the "Facets" category of EDM properties), etc.

[Model-level settings](#) can suppress some of the attributes for **all properties** in the model. You also can suppress attribute generation for each entity and complex type property **individually** by opening the **Attributes to Suppress** collection and setting any of them to *True*.

The attributes you can suppress are:

Attribute	Description
Bindable	Suggests to the UI how the property should be bound: <i>OneWay</i> , <i>TwoWay</i> , or <i>None</i> =not at all.
DefaultValue	Specifies the default value to use during new entity initialization .
DisplayName	Suggests the text that the UI should use to write labels and grid column headers.
Editable	Suggests to the UI whether it should enable editing of this property.
Key	Whether this property participates in the EntityKey .

Metadata	Whether to look at the entity's companion metadata class for the attributes to adorn the property.
ReadOnly	Whether the property is read-only or read/write.
Validation	Whether to generate validation attributes .

Remember: these settings enable or suppress **attribute generation**. They **do not set the values** of the attributes. For example, setting *Editable* false suppresses the *EditableAttribute*; it does not disable editability.

Finally, for ultimate control over attribute generation, you can [write an entity metadata class](#) and set every attribute manually.

Concurrency strategy

The "Concurrency strategy" choices are described in a [topic dedicated to concurrency](#). The choice you make here appears in the generated property code as one of the parameters to the property's *DataEntityProperty* constructor, as seen in this example:

```
using IbEm = IdeaBlade.EntityModel;
...
// RowVersion's DataEntityProperty
public static readonly IbEm.DataEntityProperty<Customer, Nullable<int>> RowVersion
    = new IbEm.DataEntityProperty<Customer, Nullable<int>>
        ("RowVersion", ..., ..., IbEm.ConcurrencyStrategy.AutoIncrement, ...);

Imports IbEm = IdeaBlade.EntityModel
...
' RowVersion's DataEntityProperty
Public Shared ReadOnly RowVersion As New _
    IbEm.DataEntityProperty(Of Customer, Nullable(Of Integer))
        ("RowVersion", ... , ..., IbEm.ConcurrencyStrategy.AutoIncrement, ...)
```

Entity property accessibility

The property getters and setters are public by default. You can choose the access mode - *Public*, *Internal*, *Protected*, or *Private* - to control the visibility of the property outside the class.

All four choices are available ... except in Silverlight. Silverlight prohibits non-public reflection; that prohibition prevents DevForce from getting and setting entity data while communicating with the server during queries and saves.

You can choose *Internal*, thus limiting access to classes defined within the model project and to assemblies that are friends of the model project. You must also make the project's internals visible to .NET assemblies.

See the "[Change member visibility](#)" topic for details.