Contents

When you **get or set an entity property**, DevForce includes a pipeline of actions which you can intercept.

When you created your model you specified the names of entity properties and their attributes, such as visibility. The generated code reflects your choices. Let's examine the generated property in more depth.

Data properties

The *DataEntityProperty* represents either a "simple" property, one in which the data type is a simple primitive such as *int* or *string*, or a "complex" property, a *ComplexObject* containing other simple and complex properties. For example:

```
public string CompanyName {
 get { return PropertyMetadata.CompanyName.GetValue(this); }
 set { PropertyMetadata.CompanyName.SetValue(this, value); }
}
```

```
Public Property CompanyName() As String
 Get
   Return PropertyMetadata.CompanyName.GetValue(Me)
 End Get
 Set(ByVal value As String)
    PropertyMetadata.CompanyName.SetValue(Me, value)
 End Set
End Property
```

The salient point is that in place of a simple backing field or automatic property we see *GetValue* and *SetValue* calls on a metadata property.

We saw in the model topic how a PropertyMetadata class is generated into each entity class to hold static definitions of all properties of the entity:

```
public static readonly IbEm.DataEntityProperty<Customer, string> CompanyName =
 new IbEm.DataEntityProperty<Customer, string>(
 "CompanyName", false, false, IbEm.ConcurrencyStrategy.None, false);
```

```
Public Shared ReadOnly CompanyName As New IbEm.DataEntityProperty(Of _
  Customer, String)("CompanyName", False, False, IbEm.ConcurrencyStrategy.None, False)
```

It's the *DataEntityProperty* which controls how the get or set is performed. When the *DataEntityProperty* is constructed, it populates its *GetterInterceptor* and *SetterInterceptor*, which control the get and set pipelines respectively. You won't often work directly with these interceptors, but you can both interrogate the interceptor to see its actions, and add and remove actions too.

Get pipeline

- If property interception is enabled for the *EntityGroup*, which it is by default, then all appropriate *BeforeGet* property interceptors are invoked. These property interceptors may cancel the get altogether.
- The default version of the property value is retrieved.
- If property interception is enabled, all *AfterGet* property interceptors are called. These interceptors may modify the retrieved value (without modifying the backing store), or cancel the get.

You can use *get* property interceptors to audit access to properties, to cancel access, and to modfiy the retrieved value. Don't rely on property interceptors as a robust security mechanism though.

Set pipeline

- If property interception is enabled for the *EntityGroup*, which it is by default, then all *BeforeSet* property interceptors are invoked. These property interceptors may modify the value, and cancel the set altogether.

- *BeforeSet* verifiers are called. If verifier options specify *ShouldExitOnBeforeSetError* then the set does not continue if a validation error occurred.
- A check is made for the equality of the old and new values. If equivalent, the property set does not take place.
- The *EntityGroup.EntityChanging* event is fired. If the change is cancelled further set processing stops.
- The *EntityGroup.EntityPropertyChanging* event is fired. If the change is cancelled further set processing stops.
- The new value is set in the *CurrentValues* store for the entity.
- The *PropertyChanged* event is fired.
- The *EntityGroup.EntityPropertyChanged* event is fired.
- The *EntityGroup.EntityChanged* event is fired.
- *AfterSet* verifiers are called.
- If property interception is enabled for the *EntityGroup*, all *AfterSet* property interceptors are invoked.

As you can see, there are a number of ways you can fine-tune this process. You'll most commonly use property interceptors to inspect and possibly modify the value to be set, and validation to ensure the correctness of the data per your validation rules.

Navigation properties

The *NavigationEntityProperty* represents either a related entity, called a scalar reference, or a list of entities, called a list reference.

By default, all navigation properties are lazily loaded. In Silverlight, they're loaded using asynchronous navigation. Your first get of a property value will cause the related entity or entities to be lazily loaded into the entity cache.

You can find a full discussion of navigation properties and how they are loaded in the query topic. Here we'll briefly discuss the get and set pipelines for scalar and list navigation properties.

Scalar navigation properties

Here's the now familiar generated property:

```
public Customer Customer {
 get { return PropertyMetadata.Customer.GetValue(this); }
 set { PropertyMetadata.Customer.SetValue(this, value); }
}
```

```
Public Property CompanyName() As String
 Get
   Return PropertyMetadata.CompanyName.GetValue(Me)
 End Get
 Set(ByVal value As String)
    PropertyMetadata.CompanyName.SetValue(Me, value)
 End Set
End Property
```

... and the corresponding *NavigationScalarEntityProperty*:

```
public static readonly IbEm.NavigationScalarEntityProperty<OrderSummary, Customer> Customer =
    new IbEm.NavigationScalarEntityProperty<OrderSummary, Customer>(
   "Customer", true, "FK_OrderSummary_Customer", IbEm.QueryDirection.ToRole2);
```

```
Public Shared ReadOnly Customer As New IbEm.NavigationScalarEntityProperty(Of OrderSummary, _
  Customer)("Customer", True, "FK_OrderSummary_Customer", IbEm.QueryDirection.ToRole2)
```

Like a simple property, a *NavigationScalarEntityProeprty* will have both a *GetterInterceptor* and *SetterInterceptor*. You can inspect and add property interceptor actions to these interceptors.

Get pipeline

- If property interception is enabled for the *EntityGroup*, which it is by default, then all appropriate *BeforeGet* property interceptors are invoked. These property interceptors may cancel the get altogether.
- Determine if a query should be issued:
  - If the current *FetchStrategy* is *CacheOnly* or the *EntityReferenceLoadStrategy* is *DoNotLoad*, then the current *ScalarEntityReference* value is returned. This might be a null entity.
  - If the referenced entity has already been loaded it is returned.
  - Otherwise either an *EntityKeyQuery* or *EntityRelationQuery* is built to retrieve the related entity. An *EntityKeyQuery* is used when navigating to a "principal" entity if all *EntityKey* values are present.
  - The query is executed, synchronously or asynchronously depending on the setting of *UseAsyncNavigation*. If synchronous, the retrieved entity is returned from the get; if asynchronous, a pending entity is returned.
- If property interception is enabled all appropriate *AfterGet* property interceptors are called.

Set pipeline

- If property interception is enabled for the *EntityGroup*, which it is by default, then all appropriate *BeforeSet* property interceptors are invoked. These property interceptors may cancel the set altogether.
- *BeforeSet* verifiers are called. If verifier options specify *ShouldExitOnBeforeSetError* then the set does not continue if a validation error occurred.
- A check is made for the equality of the old and new values. If equivalent, the property set does not take place.
- The *EntityGroup.EntityChanging* event is fired. If the change is cancelled further set processing stops.
- The *EntityGroup.EntityPropertyChanging* event is fired for the corresponding foreign key property. If the change is cancelled further set processing stops.
- The new scalar reference value is set.
- The *PropertyChanged* event is fired for the foreign key property.
- The *EntityGroup.EntityPropertyChanged* event is fired for the foreign key property.
- The *PropertyChanged* event is fired for the navigation property.
- The *EntityGroup.EntityPropertyChanged* event is fired for the navigation property.
- The *EntityGroup.EntityChanged* event is fired.
- *AfterSet* verifiers are called.
- If property interception is enabled all appropriate *AfterSet* property interceptors are called.

The set pipeline for a reference property is more complex than for an ordinary data property. Because both the navigation property reference and the corresponding foreign key property are set, there are more opportunities for interception and validation, and more events are raised. There is actually quite a bit more going on under the hood, such as "fixup" of keys and attaching of entities, but we won't discuss that here.

List navigation properties

Our generated property:

```
public IbEm.RelatedEntityList<Order> Orders {
 get { return PropertyMetadata.Orders.GetValue(this); }
}
```

```
Public ReadOnly Property Orders() As IbEm.RelatedEntityList(Of Order)
 Get
  Return PropertyMetadata.Orders.GetValue(Me)
 End Get
End Property
```

We see there's no setter with a *NavigationListEntityProperty*, since you retrieve the list (even if empty) and add and remove entities to the list.

Here's the *NavigationListEntityProperty*:

```
public static readonly IbEm.NavigationListEntityProperty<Customer, Order> Orders =
 new IbEm.NavigationListEntityProperty<Customer, Order>(
 "Orders", true, "FK_Order_Customer", IbEm.QueryDirection.ToRole1);
```

```
Public Shared ReadOnly Orders As New IbEm.NavigationListEntityProperty(Of Customer, Order)("Orders", _
 True, "FK_Order_Customer", IbEm.QueryDirection.ToRole1)
```

The *NavigationListEntityProperty* will have only a *GetterInterceptor*. You can inspect and add property interceptor actions to this interceptor.

Get pipeline

- If property interception is enabled for the *EntityGroup*, which it is by default, then all appropriate *BeforeGet* property interceptors are invoked. These property interceptors may cancel the get altogether.
- Determine if a query should be issued:
  - If the current *FetchStrategy* is *CacheOnly* or the *EntityReferenceLoadStrategy* is *DoNotLoad*, then the current *ListEntityReference* values are returned. This might be an empty list.
  - If the referenced list has already been loaded it is returned.
  - Otherwise an *EntityRelationQuery* is built to retrieve the related entities.
  - The query is executed, synchronously or asynchronously depending on the setting of *UseAsyncNavigation*. If synchronous, the retrieved list of entities is returned from the get; if asynchronous, a pending entity list is returned.
- If property interception is enabled all appropriate *AfterGet* property interceptors are called.

Add and remove

If a collection is not read only you can add and remove related entities. A *RelatedEntityList* will be read only if created from a *NavigationListEntityProperty* having the *IsCollectionReadOnly* flag set, or if the flag has been set directly on the list.

The pipeline for adds and removes is similar to that of a scalar navigation property setter, since the corresponding scalar setter is what's called under the hood. Many-to-many navigation properties are the exception, however, since adds and removes in a many-to-many relationship target relationships instead of entities.