Contents

Any time a navigation property is accessed, DevForce needs to determine whether to try to fetch the related entity or entities from the backend datastore or to use previously retrieved data. In general, the first time any navigation property on an entity is accessed, data will be retrieved from the database and all subsequent attempts to access the property will use the local cache. **Navigation properties and data retrieval** can be modified in several ways.

Navigation properties

By default, all navigation properties are lazily loaded. To change this "load strategy" you can do so easily using the *ReferenceStrategy* defined for every NavigationEntityProperty.

The *EntityReferenceLoadStrategy* is an enum which allows you to choose among "Lazy", "DoNotLoad", and "Load" options. The default, "Lazy" indicates that the related data will be loaded once when first accessed. Use "DoNotLoad" if the related data should not ever be retrieved from the back end data store; and use "Load" to force a retrieval from the back end data store upon every access.

You can set the strategy for a specific property, or for the *EntityManager* as a whole.

Set the load strategy for a specific property

Setting a reference strategy at the property level is a global setting for this property across all entity instances and EntityManagers. This setting overrides the EntityManager's DefaultEntityReferenceStrategy for this property.

For example, the following will reset the strategy for the Customer.OrderSummaries property to always load from the database when accessed:

```
var navProp = Customer.PropertyMetadata.OrderSummaries;
var ers = new EntityReferenceStrategy(EntityReferenceLoadStrategy.Load, MergeStrategy.PreserveChanges);
navProp.ReferenceStrategy = ers;
```

Set the load strategy for an EntityManager

Set the *DefaultEntityReferenceStrategy* to change the load strategy for an *EntityManager*.

For example, to force every access of any navigation property to go to the database and to have the results of that navigation overwrite any local changes:

```
myEntityManager.DefaultEntityReferenceStrategy =
 new EntityReferenceStrategy(EntityReferenceLoadStrategy.Load,
   MergeStrategy.OverwriteChanges);
```

```
MyEntityManager.DefaultEntityReferenceStrategy = _
 New EntityReferenceStrategy(EntityReferenceLoadStrategy.Load, _
   MergeStrategy.OverwriteChanges)
```

Components of an EntityReferenceStrategy

As shown in the examples above, a *ReferenceStrategy* is itself is made up of a *EntityReferenceLoadStrategy* and a *MergeStrategy*. The *EntityReferenceLoadStrategy* determines under what conditions data should be 'loaded' (i.e. retrieved from the database) and the *MergeStrategy* determines how that data, if loaded, gets merged into the local entity cache.

LoadStrategy	Meaning
Lazy	first access to the property should go to the database; subsequent accesses should use the local cache
Load	for every access always try to load from the database

DoNotLoad	never try to load from the database; property access always looks in cache. You can load the entities manually as described below.

The default *LoadStrategy* is *Lazy*; the default *MergeStrategy* is *PreserveChanges*. These concepts are also discussed under the Entity cache and Entity metadata topics.

Asynchronous navigations

There is actually a third part of every EntityReferenceStrategy that determines whether the related property navigation should be performed asynchronously or not. This is represented by the *IsAsync* property and is usually defaulted based on the runtime environment; in Silverlight and mobile environments *IsAsync=true* and in the Desktop CLR *IsAsync=false*. However, there is a seperate constructor overload that will allow you to set the IsAsync flag to something other than the default. It's signature is the same as that shown above with one additional final boolean parameter.

Resetting the default EntityReferenceStrategy

As mentioned earlier, DevForce provides a way of specifying a 'default' EntityReferenceStrategy via the *DefaultEntityReferenceStrategy*. This is useful because it is very common for applications to have a 'preferred' model for property navigation. Every EntityReference associated with an entity within an EntityManager will use this EntityReferenceStrategy by default. Specifying an EntityReferenceStrategy for a navigation property overrides this default, as was shown above. Resetting a navigation property to use the default is accomplished by setting NavigationProperty's ReferenceStrategy to null.

```
Customer.PropertyMetadata.Orders.ReferenceStrategy = null;
```

```
Customer.PropertyMetadata.Orders.ReferenceStrategy = Nothing
```

EntityReference

Every navigation property on an entity has a corresponding *EntityReferenceBase*. This *EntityReferenceBase* can be obtained by calling *NavigationEntityProperty.GetEntityReference* method. Depending on whether the navigation property returns a scalar or a list, the actual EntityReference returned will be either a *ScalarEntityReference<T>* or a *ListEntityReference<T>*.

You don't usually need to work with the *EntityReference*, but if you do, it's easy to access::

```
var listref = Customer.PropertyMetadata.Orders.GetEntityReference(myCustomer);
```

Using the *EntityReference* you can check whether related entities have been loaded using *IsLoaded*, or force a reload with *Load*.

Loaded state

The *ReferenceStrategy* governs the behavior of a navigation property for *all* instances of the entity type. You can tweak the current state of a *particular* entity's navigation property. For example, you can force the "cust" *Customer* entity to act as if its orders have already been loaded into cache:

```
Customer.PropertyMetadata.Orders
      .GetEntityReference(cust)
      .IsLoaded = true;
```

```
Customer.PropertyMetadata.Orders _
      .GetEntityReference(cust) _
      .IsLoaded = True
```

If you didn't know the entity type or property name at compile time, you could write a general function to do this. The pertinent statement would be something like:

```
anEntity.EntityAspect.EntityMetadata
      .NavigationProperties
      .First(p => p.Name == somePropertyName)
      .GetEntityReference(anEntity)
      .IsLoaded = true;
```

```
anEntity.EntityAspect.EntityMetadata _
      .NavigationProperties _
      .First(Function(p) p.Name = somePropertyName) _
      .GetEntityReference(anEntity) _
      .IsLoaded = True
```

This statement would have the same effect as before if *anEntity* == *cust* and *somePropertyName* == "Orders".

Force loading

Sometimes you want to force a navigation property to refresh. Here's one way to force a (re)load of the example customer's orders:

```
Customer.PropertyMetadata.Orders
        .GetEntityReference(cust)
        .Load(MergeStrategy.PreserveChanges);
```

```
Customer.PropertyMetadata.Orders _
        .GetEntityReference(cust) _
        .Load(MergeStrategy.PreserveChanges)
```

The navigation property load operation runs synchronously in Desktop and web apps, and asynchronously in other environments.