Contents

There are several ways of **forcing the requery** of previously cached data. This can be important when you want to insure that the local entity cache has the absolutely latest data.

One approach is to use one of the *RefetchEntities* overloads on the *EntityManager*:

RefetchEntity / RefetchEntities methods

```
public void RefetchEntity(Object entity, MergeStrategy mergeStrategy);
public void RefetchEntities(IEnumerable entities, MergeStrategy mergeStrategy);
public void RefetchEntities(EntityKeyList entityKeys, MergeStrategy mergeStrategy);
public void RefetchEntities(EntityState entityState, MergeStrategy mergeStrategy);
```

```
Public Sub RefetchEntity(ByVal entity As Object, ByVal mergeStrategy As MergeStrategy)
Public Sub RefetchEntities(ByVal entities As IEnumerable, ByVal mergeStrategy As MergeStrategy)
Public Sub RefetchEntities(ByVal entityKeys As EntityKeyList, ByVal mergeStrategy As MergeStrategy)
Public Sub RefetchEntities(ByVal entityState As EntityState, ByVal mergeStrategy As MergeStrategy)
```

There are also asynchronous versions of each of the above.

One warning about these methods: they are all designed to work with reasonably small numbers of entities (< 1000). While they will work with larger numbers of entities, performance may be poor. The reason for this is that these methods are implemented so that they in effect create a large "IN" or "OR" query for all of the desired entities by key. The query expression itself can therefore become very large for large numbers of entities. Expressions that are this large will have performance impacts in both serialization as well as query compilation. For those cases where very large numbers of entities need to be refreshed, it is usually a better idea to write a "covering" query that is much smaller textually but returns approximately the same results. You may find that even though you return more entities than are needed with this covering query, the resulting overall performance is still better.

The *MergeStrategy* determines how the refetched data will be merged into cache. *OverwriteChanges* replaces the cached entities, overwriting our pending changes. We often want to (a) keep pending changes but (b) refresh copies of unmodified entities. The *PreserveChanges…* strategies can help us achieve our purpose.

Strategy	Description
OverwriteChanges	Overwrites the cached entity with incoming data and uses the EntityState of the incoming entity
PreserveChanges	Replace unchanged entities but keep changed entities as they are.
PreserveChangesUnlessOriginalObsolete	Preserves the persistent state of the cached entity if the entity is current. Overwrites an entity if it is EntityState of the incoming entity.
PreserveChangesUpdateOriginal	Preserves the persistent state of the cached entity whether it is current or not. Overwrites the Origi obsolete.

How is *Current* determined? DevForce uses the *ConcurrencyProperties* you've defined for the Entity. If the values of the concurrency properties are the same between the original and incoming entity, then the entity is considered current. If you haven't defined a concurrency property for the entity then DevForce has no way of determining currency correctly, so assumes that the entity is current.

What do we mean by the *Original* version? Every entity can have up to three "versions" of property values. All entities will have "current" values; modified entities will also have "original" values, i.e., the property values before the entity was modified. Entities in edit mode will also have "proposed" values, i.e., the values changed during the edit but not yet committed.

Merge strategies are described in further detail here.

Fetch and Re-fetch of navigation properties

We saw in the Navigation properties and data retrieval topic how to obtain the *NavigationEntityProperty* and corresponding *EntityReference* for related entities.

By default navigation properties are lazily loaded upon first access. You can change the *EntityReferenceLoadStrategy* to instead force the property to be reloaded upon every access by changing the strategy to *Load*.

You can also reload a navigation property as needed using either the *Reload* method on the *RelatedEntityList* or the *Load* method on the *EntityReference*. They are equivalent.

```
anOrder.OrderDetails.Reload(MergeStrategy.OverwriteChanges);
```

```
Order.PropertyMetadata.OrderDetails.GetEntityReference(order).Load(MergeStrategy.OverwriteChanges);
```

The MergeStrategy here functions exactly the same as when used with the *EntityManager RefetchEntities* methods described above.