Contents

Modify entities in the entity cache.

Overview

Modifications to your entities in cache is a straightforward process. You've loaded entities into the EntityManager cache - usually by querying for them, but also by creating and saving new entities, importing entities from another *EntityManager*, and by loading a previously saved snapshot.

However the entities got there, you can then modify them as needed. These modifications will often be through your UI via data binding to the properties of your entities, allowing user changes to be pushed into the properties.

To persist your changes to the data store, you'll call one of the SaveChanges overloads on the *EntityManager*.

The generated setter

Most properties of your entities have public getters and setters. You've set the access in the EDM Designer, and the generated code has dutifully carried out your wishes:

```
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
}
```

When the property is data bound, both its getter and setter will be invoked to retrieve and set the value.

To change a property value, you'll generally use the property setter,

```
myCustomer.CompanyName = "Emeryville Eats";
```

```
myCustomer.CompanyName = "Emeryville Eats"
```

.

The first modification to an unchanged entity causes its *EntityState* to become *Modified*. If you're working with a newly created entity, its *EntityState* will remain as *Added*. Use the EntityAspect for the entity to see its current state.

Every time a property is modified DevForce will automatically raise a *PropertyChanged* event for the specific property name. Data bound controls will listen for these events to refresh the display.

We saw the set pipeline in the "get and set a property" topic. Along with *BeforeSet* and *AfterSet* property interception, validation may be performed, and a number of events are raised in addition to *PropertyChanged*.

Miscellaneous topics

Thread safety

Like the *EntityManager*, an entity is not thread-safe. Unlike the *EntityManager*, DevForce won't try to detect that the entity is being used on multiple threads. So if you decide to have a background worker massage your entities while the application user is editing them in the UI, you're writing your own recipe for disaster. If you need to use an *EntityManager* and its entities across threads, the best thing to do is import the entities into another manager, so that each thread has its own *EntityManager* to work with.

Modifying EntityKey properties

Once at entity has been created and saved, you cannot modify its *EntityKey*. You'll receive an exception when saving the change if you try. If you do find you need to modify the key, you can clone the original entity, set the new *EntityKey* and add the clone to the EntityManager, and delete the original entity.

All entities implement ICloneable via an explicit interface implementation, so you must first cast the entity to *ICloneable*, for example:

```
Customer customerCopy = ((ICloneable)aCustomer).Clone() as Customer;
```

```
Customer customerCopy = ((ICloneable)aCustomer).Clone() as Customer
```

Changing concurrency properties

Concurrency property values are often set by the data store - for example via a database trigger or a timestamp column - or with the DevForce "auto" settings. But you can also set the new concurrency value in your application code. Just remember to set the *ConcurrencyStrategy* to *Client*, and you can then set the property value as wanted, for example:

```
aCustomer.LastChangeDt = DateTime.Now.ToUniversalTime();
```

```
aCustomer.LastChangeDt = DateTime.Now.ToUniversalTime()
```