Contents

An entity's *EntityState* tells you if it is attached to an *EntityManager*, if it has pending (unsaved) changes, whether it was newly created, previously existing, or scheduled for deletion. This topic covers what these states mean, how they change, and some ways to program with EntityState.

Has it changed?

No direct property of *Customer* can tell you if a particular *Customer* instance has been changed. That's not a question about "Customer-ness"; it's an entity infrastructure question. You access a *Customer* object's entity internals by means of its EntityAspect property. **EntityAspect** opens the door to a wealth of information about your entity.

Three EntityAspect members can tell you if the entity has changed:

- *HasChanges*
- *IsChanged*
- *EntityState*
  HasChanges() and *IsChanged* are synonyms. One is a method; the other is a property to which you can data bind. *IsChanged* delegates to *HasChanges()*. Use either like this

```
if (aCustomer.EntityAspect.IsChanged) // do something
```

```
If (aCustomer.EntityAspect.IsChanged) Then  ' do something
```

HasChanges() is defined in terms of the *EntityState*. It reports that the entity has changes if the entity's *EntityState* is **any value other than *Unchanged***.

A thought for later: *IsChanged* is always true for a *Detached* entity whether or not it actually had changes before it was detached. In other words, don't use *HasChanges()* and *IsChanged* with detached entities.

HasChanges() and *IsChanged* are syntactic sugar wrapped around the property that really matter, the ***EntityState***

Introduction to EntityState

EntityState addresses these three questions:

- Is the entity attached to an EntityManager or is it detached?
- Is the entity changed or not
- If changed, is it an added entity, a modified entity, or an entity marked for deletion?

The answer is one of the *IdeaBlade.EntityModel.**EntityState*** enums returned by the *EntityAspect.EntityState* property.

```
currentState = aCustomer.EntityAspect.EntityState; // an EntityState enum
```

```
currentState = aCustomer.EntityAspect.EntityState ' an EntityState enum
```

The currentState will be one of these *EntityState enums* :

EntityState	Summary
Detached	The entity is not attached to an EntityManager. It is in this state upon creation or after being removed from an EntityManager.
Unchanged	The entity is attached to the EntityManager, presumed to exist in the database, and has not changed since last queried or saved.
Added	The entity has been added to the EntityManager and is presumed to be new (not in the database).
Modified	The entity is attached to the EntityManager, presumed to exist in the database, and has been modified since last queried or saved.
Deleted	The entity was deleted using the Entity.Delete() method.

In this topic we discuss what the states mean and how to use them. A separate topic covers the operations that changea an entity's EntityState.

The Detached EntityState

You usually work with an entity that is attached to an EntityManager. An **attached** entity is an entity that is in an EntityManager's entity cache. You can ask it for its EntityManager:

```
manager = aCustomer.EntityAspect.EntityManager;
```
```
manager = aCustomer.EntityAspect.EntityManager
```

The variable *manager* has a value if the entity is attached; is is null if the entity is not attached.

It's easier to check the EntityState than to test for a null EntityManager.
An entity is attached if its EntityState is not *Detached*.

```
if (aCustomer.EntityAspect.EntityState != EntityState.Detached) // it's attached
```
```
If (aCustomer.EntityAspect.EntityState <> EntityState.Detached) Then ' it's attached
```

There are reasons to work with a detached entity and certain operations automatically detach an entity. Another topic covers attaching and detaching entities in more detail. Here we discuss some of the implications of being attached or detached.

Certain infrastructural features only work when the entity is attached. For example, you can only get to the database via a navigation property when the entity is attached.

Consider the following statement:

```
aCustomer = anOrder.Customer; // get the Order's parent Customer
```
```
aCustomer = anOrder.Customer ' get the Order's parent Customer
```

The property behaves differently for attached and detached entities.

If *anOrder* is attached, its *aCustomer* is either a real *Customer* entity or the special form of a *Customer* entity called "the **null entity**" (aka "the **nullo**") depending upon whether *anOrder* has a customer or not. If DevForce isn't sure, it could query the database for the parent Customer.

You can check if the Order's Customer is real or the nullo.

```
if (aCustomer.EntityAspect.IsNullEntity) // ...
```
```
If (aCustomer.EntityAspect.IsNullEntity) Then ' ...
```

If *anOrder* is detached, *aCustomer* is probably null. The *anOrder* object doesn't have an EntityManager so it can't find its parent *Customer* in cache and it has no way query the database. It might have a Customer, left over from its former life as an attached entity. But it probably doesn't.

That means that the example above would throw a *NullReferenceException*.

Don't expect automatic validation when you set a property of a *Detached* entity. Automatic property validation depends upon access to the *EntityAspect.**VerifierEngine***. The *VerifierEngine* is null for a *Detached* so the data property skips the validation step.

Make sure you know when you're working with a *Detached* entity.

Use combo-EntityStates to find entities in cache

EntityState is defined as a flag enum which means individual enum values can be OR'd together to represent a combination of states.

```
addedOrModified = EntityState.Added | EntityState.Modified;
```
```
addedOrModified = EntityState.Added Or EntityState.Modified
```

The EntityState enum includes two particularly useful combinations:

EntityState	Summary
AnyAddedModifiedOrDeleted	Added or Modified or Deleted.
AllButDetached	All states except detached.

You can use these combo enums to select entities from the cache.

```
// All entities in cache with pending changes
changedEntities =  manager.FindEntities(EntityState.AnyAddedModifiedOrDeleted);
// All Customer entities in cache
custsInCache =  manager.FindEntities<Customer>(AllButDetached);
```

```
' All entities in cache with pending changes
changedEntities =  manager.FindEntities(EntityState.AnyAddedModifiedOrDeleted)
' All Customer entities in cache
custsInCache =  manager.FindEntities(Of Customer)(AllButDetached)
```

Simplify EntityState checking with extension methods

You frequently ask if an entity is in a particular state or one of a common set of states. The code to get the answer is not difficult but it is tedious.

```
 currentState = aCustomer.EntityAspect.EntityState;
if (EntityState.Deleted == currentState) // ...
if ((currentState & (EntityState.Added | EntityState.Modified)) > 0) // ...
```

```
 currentState = aCustomer.EntityAspect.EntityState
If (EntityState.Deleted = currentState) Then ' ...
If ((currentState And (EntityState.Added Or EntityState.Modified)) > 0) Then ' ...
```

These boolean EntityState extension methods make it easier to check for specific states and combination.

- *IsDetached()*
- *IsModified()*
- *IsDeleted()*
- *IsUnchanged()*
- *IsAddedOrModified()*
- *IsAddedOrModifiedOrDeleted()*
- *IsDeletedOrDetached()*
- *IsDeletedOrModified()*

Here are the same state-checking statements as above, rewritten with extension methods.

```
 currentState = aCustomer.EntityAspect.EntityState;
if (currentState.IsDeleted()) // ...
if ((currentState.IsAddedOrModified()) // ...
```

```
 currentState = aCustomer.EntityAspect.EntityState
If currentState.IsDeleted() Then '...
If (currentState.IsAddedOrModified() Then '...
```