Contents

The **entity cache** is an in-memory container of entity objects that belong to a specific *EntityManager*. Working with entities in cache requires little effort on your part. Most queries automatically put retrieved entities into the cache, and when you ask the *EntityManager* to save, it saves the changed entities in cache to the datastore.

DevForce automatically decides if subsequent queries can be safely satisfied from cache, and if so, this saves a trip to the server and improves performance. However, you can always refresh entities in cache if you need to.

Although you will typically work with entities that reside in an *EntityManager's* cache, you *can* work with entities that are not yet in cache and entities that were once in cache but have since been detached - but that's unusual. Many of the important features of entities, including the ability to navigate to related entities and to save changes, are only available when the entities reside in cache.

Entities in cache are unique

An entity in cache is identified by its EntityKey and type (e.g. *Customer*). All entities in cache have unique *EntityKeys*; a cache can't have two instances of *Customer* with the same *CustomerID*. Of course no one can prevent you from creating two *Customer* objects with the same *CustomerID*. But they can't both be in the same *EntityManager* cache at the same time.

This is true for deleted entities as well. They may seem invisible but they are there, in cache, until they are saved. Only after they have been saved successfully do they depart the cache.

Add entities to cache

Entities usually enter the cache as a by-product of a query. This happens automatically without any action on your part (unless you specifically want to change the behavior).

You can also add entities to cache explictly in three ways:

1. By adding it to an *EntityManager* with the *AddEntity* method. An added entity is regarded as a new entity that will be inserted into the database if saved.
2. By attaching it to an *EntityManager* with the *AttachEntity* method. An attached entity is regarded as an existing, unmodified entity like one that has been queried.
3. By importing it with the *EntityManager's ImportEntities* method.

Remove entities from cache

Deleting an existing entity doesn't remove it from cache but saving a deleted entity does. The save removes the deleted entity implicitly. You can remove an entity from cache explicitly as well:

```
manager.RemoveEntity(someCustomer);
```

```
manager.RemoveEntity(someCustomer)
```

Call the *EntityManager's Clear* method to remove *every* entity from the cache.

Is the entity in cache or not?

In the code above, *someCustomer* is a reference to a *Customer* entity. They entity didn't disappear because we removed it from cache. It's still an entity. It's now a *Detached* entity.

We can ask the entity if it is in cache or *Detached* by inquiring about its entitystate. The following line verifies that *someCustomer* was detached after we removed it.

```
Assert.IsTrue(someCustomer.EntityAspect.EntityState.IsDetached());
```

```
Assert.IsTrue(someCustomer.EntityAspect.EntityState.IsDetached())
```

EntityState can tell us more than whether the entity is attached or detached.

Find entities in cache

You can explicitly query the cache by using a *CacheOnly* *QueryStrategy* .

You can also use *FindEntities* or *FindEntityGraph* to search the cache.

```
var foundCust = manager.FindEntity(cust1.EntityAspect.EntityKey);
Assert.AreSame(cust1, foundCust);
```

```
Dim foundCust = manager.FindEntity(cust1.EntityAspect.EntityKey)
Assert.AreSame(cust1, foundCust)
```

Unlike a query, you can find an entity in cache even if it is deleted.

```
cust1.EntityAspect.Delete(); // marked for deletion
var foundCust =
    manager.FindEntity(cust1.EntityAspect.EntityKey, includeDeleted:true );
Assert.AreSame(cust1, foundCust);
```

```
cust1.EntityAspect.Delete() ' marked for deletion
Dim foundCust =
    manager.FindEntity(cust1.EntityAspect.EntityKey, includeDeleted:=True )
Assert.AreSame(cust1, foundCust)
```

You can also find entities in cache using LINQ for objects:

```
var foundCust = manager
    .FindEntities(EntityState.AllButDetached)
    .OfType<Customer>()
    .Where(c => c.CompanyName == "Acme") // cust1's name
   .FirstOrDefault();
Assert.AreSame(cust1, foundCust);
```

```
Dim foundCust = manager _
   .FindEntities(EntityState.AllButDetached) _
   .OfType(Of Customer)() _
   .Where(Function(c) c.CompanyName = "Acme") _
   .FirstOrDefault()
Assert.AreSame(cust1, foundCust)
```

Notice that

- *FindEntities* filters by entitystate; here we ask for all cached states (*AllButDetached*).
- *FindEntities* returns an *IEnumerable*; we cast it to *IEnumerable* of *Customer* in order to query it with LINQ.
- *FindEntities* won't return any *Detached* entities because *Detached* entities are not associated with an *EntityManager*.

Export cache as an EntityCacheState

The *EntityManager* itself is not serializable. But its cache contents are serializable when in the form of an *EntityCacheState* . An *EntityCacheState* is a snapshot of the entities in cache. This snapshot can be handed around inside the client application, serialized to file, restored from file, even sent to the server as a parameter in a remote server method call.

To get an *EntityCacheState*, start with the manager's *CacheStateManager* . Its *GetCacheState* method can return an *EntityCacheState* with all or only some of the entities in cache.

In the following unrealistic example, we use an *EntityCacheState* to copy a cache from one manager to another:

```
// Get EntityCacheState with all cached entities using the CacheStateManager
var ecs = manager1.CacheStateManager.GetCacheState();
// Create 2nd manager
var manager2 = new EntityManager(shouldConnect: false);
// "Restore" into manager2 with the contents of the ECS from manager1
manager2.CacheStateManager.RestoreCacheState(ecs);
// Prove that manager2 has a customer with same ID as cust1
var foundCust = manager2.FindEntity(cust1.EntityAspect.EntityKey);
// But the foundCust is not the same as cust1
// because cust1 still belongs to manager1
Assert.AreNotSame(cust1, foundCust);
```

```
' Get EntityCacheState with all cached entities using the CacheStateManager
```

```
Dim ecs = manager1.CacheStateManager.GetCacheState()
' Create 2nd manager
Dim manager2 = New EntityManager(shouldConnect:= False)
' "Restore" into manager2 with the contents of the ECS from manager1
manager2.CacheStateManager.RestoreCacheState(ecs)
' Prove that manager2 has a customer with same ID as cust1
Dim foundCust = manager2.FindEntity(cust1.EntityAspect.EntityKey)
' But the foundCust is not the same as cust1
' because cust1 still belongs to manager1
Assert.AreNotSame(cust1, foundCust)
```

A *CacheStateManager* can also save or restore an *EntityCacheState* from a file.

Entity cache structure

You rarely need to probe around inside the entity cache itself. Most of the time, the cache works transparently and if you need to be explicit, the cache-only query and the *Find* methods are the preferred ways to retrieve entities from cache.

When you need to watch the cache for activity regarding a particular type, it helps to know about *EntityGroups* . The cache is organized a collection of *EntityGroup*s. Each group holds the cached entities for a particular type of entity.

You can discover what entity types the manager has seen by asking for its *EntityGroups*

```
manager = new EntityManager(shouldConnect:false);
Assert.AreEqual(0, manager.GetEntityGroups().Count());
```

```
manager = New EntityManager(shouldConnect:=False)
Assert.AreEqual(0, manager.GetEntityGroups().Count())
```

A new manager's cache has no groups. It acquires groups as different entity types are added to the cache or are referred to by an entity that was queried into cache.

```
manager = new EntityManager(shouldConnect:false);
manager.AddEntity(new Customer());
Assert.AreEqual(1, manager.GetEntityGroups().Count());
```

```
manager = New EntityManager(shouldConnect:=False)
manager.AddEntity(New Customer())
Assert.AreEqual(1, manager.GetEntityGroups().Count())
```

You can ask for a specific *EntityGroup*:

```
customerGroup = manager.GetEntityGroup<Customer>();
```

```
customerGroup = manager.GetEntityGroup(Of Customer)()
```

Clearing the manager's cache (manager.Clear()) removes all groups.

Listen to cache changes

The cache can tell you when an entity has been attached, changed, or detached.

The following fragment shows how to listen for any change to the cache or to cache changes for a particular entity type.

```
// Listen for all changes to cache
var cacheChanges = new List<EntityAction>();
int cacheChangeCount = 0;
manager.EntityChanged += (s, e) => cacheChanges.Add(e.Action);
// Listen for changes to customers in cache
var custChanges = new List<EntityAction>();
int custChangeCount = 0;
var custGrp = manager.GetEntityGroup<Customer>();
custGrp.EntityChanged += (s, e) => custChanges.Add(e.Action);
// Customer changes
var cust = new Customer();
manager.AddEntity(cust);          cacheChangeCount++; custChangeCount++;
cust.CompanyName = "Acme";        cacheChangeCount++; custChangeCount++;
cust.CompanyName = "Beta";        cacheChangeCount++; custChangeCount++;
cust.EntityAspect.AcceptChanges(); cacheChangeCount++; custChangeCount++;
cust.EntityAspect.Delete();       cacheChangeCount++; custChangeCount++;
// Employee change
var emp = new Employee {EmployeeID = 42};
```

```
manager.AttachEntity(emp); cacheChangeCount++;
Assert.AreEqual(cacheChangeCount, cacheChanges.Count(),
    "not all cache changes were signaled");
Assert.AreEqual(custChangeCount, custChanges.Count(),
   "not all cust changes were signaled");
Assert.IsTrue(cacheChangeCount > custChangeCount,
   "should have more cache changes than cust changes");
```

```
' Listen for all changes to cache
Dim cacheChanges = New List(Of EntityAction)()
Dim cacheChangeCount As Integer = 0
AddHandler manager.EntityChanged, Sub(s, e) cacheChanges.Add(e.Action)
' Listen for changes to customers in cache
Dim custChanges = New List(Of EntityAction)()
Dim custChangeCount As Integer = 0
Dim custGrp = manager.GetEntityGroup(Of Customer)()
AddHandler custGrp.EntityChanged, Sub(s, e) custChanges.Add(e.Action)
' Customer changes
Dim cust = New Customer()
manager.AddEntity(cust)
cacheChangeCount += 1
custChangeCount += 1
cust.CompanyName = "Acme"
cacheChangeCount += 1
custChangeCount += 1
cust.CompanyName = "Beta"
cacheChangeCount += 1
custChangeCount += 1
cust.EntityAspect.AcceptChanges()
cacheChangeCount += 1
custChangeCount += 1
cust.EntityAspect.Delete()
cacheChangeCount += 1
custChangeCount += 1
' Employee change
Dim emp = New Employee With {.EmployeeID = 42}
manager.AttachEntity(emp)
cacheChangeCount += 1
Assert.AreEqual(cacheChangeCount, cacheChanges.Count(), "not all cache changes were signaled")
Assert.AreEqual(custChangeCount, custChanges.Count(), "not all cust changes were signaled")
```