Contents

- Finding entities in cache
  - Finding an entity by its EntityKey
  - Finding entities by EntityState
  - Finding an object graph
- Synchronous queries in async environments

There are several techniques to help **query and find entities in cache**.

Finding entities in cache

The EntityManager provides several means of finding entities in its entity cache. In DevForce-speak, "find" means to perform a search of cache only without attempting to access the server or datastore.

Finding an entity by its EntityKey

If you know the *EntityKey* of an entity, you can use the *FindEntity* method to find the entity in the *EntityManager* cache. This is an efficient way both to locate the entity or to prove that the entity is not in cache.

```
Employee emp = (Employee) Manager.FindEntity(new EntityKey(typeof (Employee), 1));
```

```
Dim emp As Employee = CType(Manager.FindEntity(New EntityKey(GetType(Employee), 1)), Employee)
```

If the requested entity is not found a null (nothing) will be returned from the *FindEntity* call.

Note that the *FindEntity* call returns an object, so you must cast the returned object to the entity type expected.

You can also use *FindEntity* to find an entity with an *EntityState* of *Deleted*. This is the only means of locating **an as-yet-unsaved deleted entity.**

```
var key = new EntityKey(typeof (Employee), 1);
Employee emp = (Employee) Manager.FindEntity(key), includeDeleted: true );
```

```
Dim key = New EntityKey(GetType(Employee), 1)
Dim emp As Employee = CType(Manager.FindEntity(key), Employee), includeDeleted As Employee
```

Finding entities by *EntityState*

You can also look for entities in cache based on their EntityState. You can combine *EntityStates* too, for example to look for all entities which are either *Added* or *Modified*.

FindEntities comes in both generic and non-generic forms, and will return either an *IEnumerable* or *IEnumerable<T>*. If using the non-generic overload you can still cast the results to an *IEnumerable<T>* in order to build a composable LINQ query.

```
var allAdded = manager.FindEntities(EntityState.Added);
```

```
Dim allAdded = manager.FindEntities(EntityState.Added)
```

```
var empList= manager.FindEntities<Employee>(EntityState.Unchanged);
```

```
Dim empList = manager.FindEntities(Of Employee)(EntityState.Unchanged)
```

Using LINQ you can do more interesting things with the results:

```
var addedCustomersInUK = manager.FindEntities<Customer>(EntityState.Added).Where(c => c.Country == "UK");
// .. or ..
var addedCustomersInUK = manager.FindEntities(EntityState.Added).OfType<Customer>().Where(c => c.Country == "UK");
```

```
Dim addedCustomersInUK = manager.FindEntities(Of Customer)(EntityState.Added).Where(Function(c) c.Country = "UK")
' .. or ..
Dim addedCustomersInUK = manager.FindEntities(EntityState.Added).OfType(Of Customer)().Where(Function(c) c.Country = "UK")
```

Finding an object graph

The *FindEntityGraph* method allows you to search the entity cache for an object graph (aka entity graph) - a list of entities that are navigable from one or more root entities according to a specified graph. Working with an entity graph is particularly useful when performing a save or import and you wish to work with only the object and its dependent entities.

The *FindEntityGraph* method takes one or more "root" entities, effectively the starting point of the graph, and one or more *EntitySpans* , information about the entity relationships to be followed to build the graph. The *FindEntityGraph* method also allows you to specify the *EntityState(s)* wanted.

This sounds more complicated that it really is, so here's a simple example. Here we build an entity graph of all orders and order details for a specific employee:

```
var emp = Manager.FindEntity(new EntityKey(typeof(Employee), 1));
var span = new EntitySpan(typeof(Employee), EntityRelations.FK_Order_Employee, EntityRelations.FK_OrderDetail_Order);
var entityGraph = Manager.FindEntityGraph(new[] { emp }, new[] { span }, EntityState.AllButDetached);
```

```
Dim emp = Manager.FindEntity(New EntityKey(GetType(Employee), 1))
Dim span = New EntitySpan(GetType(Employee), EntityRelations.FK_Order_Employee, _
  EntityRelations.FK_OrderDetail_Order)
Dim entityGraph = Manager.FindEntityGraph( { emp }, { span }, EntityState.AllButDetached)
```

Here's a more complex example. The *EntityManager* must be able to "walk" each *EntitySpan*, in the case above we navigated from Employee -> Order -> OrderDetail. If a relationship can't be walked, then you must supply another *EntitySpan* for it. Here we also want to retrieve all Customers for the selected Orders. To do so we need to include another *EntitySpan*, one which walks from Employee -> Order -> Customer.

```
var emp = Manager.FindEntity(new EntityKey(typeof(Employee), 1));
var span1 = new EntitySpan(typeof(Employee), EntityRelations.FK_Order_Employee, EntityRelations.FK_OrderDetail_Order);
var span2 = new EntitySpan(typeof(Employee), EntityRelations.FK_Order_Employee, EntityRelations.FK_Order_Customer);
var entityGraph = Manager.FindEntityGraph(new[] { emp }, new[] { span1, span2 }, EntityState.AllButDetached);
```

```
Dim emp = Manager.FindEntity(New EntityKey(GetType(Employee), 1))
Dim span1 = New EntitySpan(GetType(Employee), EntityRelations.FK_Order_Employee, _
  EntityRelations.FK_OrderDetail_Order)
Dim span2 = New EntitySpan(GetType(Employee), EntityRelations.FK_Order_Employee, _
  EntityRelations.FK_Order_Customer)
Dim entityGraph = Manager.FindEntityGraph( { emp }, { span1, span2 }, EntityState.AllButDetached)
```

A few more things to note.

- The "roots" of the graph do not need to be of the same type.
- *Deleted* entities can be included in the graph.
- The "graph" is just an *IList<object>*, not a special type.
- Entities not in cache are not lazily loaded when an association is navigated.

Synchronous queries in async environments

We've already seen that any query can be given a QueryStrategy to control where it's executed. When you provide a *CacheOnly* query strategy, either explicitly on the query, by setting the *DefaultQueryStrategy* on the *EntityManager*, or by disconnecting, the *EntityManager* will run the query against the local entity cache only and not attempt to communicate with the *EntityServer*. Since no server communication is performed, you can execute *CacheOnly* queries synchronously in all environments, including Silverlight and Windows Store applications.

For example, the following will work in any environment:

```
var customers = Manager.Customers.With(QueryStrategy.CacheOnly).Execute();
```

```
Dim customers = Manager.Customers.With(QueryStrategy.CacheOnly).Execute()
```

Or alternately,

```
var customers = Manager.Customers.With(QueryStrategy.CacheOnly).ToList();
```

```
Dim customers = Manager.Customers.With(QueryStrategy.CacheOnly).ToList()
```

You can also issue scalar "immediate" execution queries against cache only without using *AsScalarAsync*. Here's a *First* method executed synchronously against cache:

```
var aCustomer = Manager.Customers.With(QueryStrategy.CacheOnly).First();
```

```
Dim aCustomer = Manager.Customers.With(QueryStrategy.CacheOnly).First()
```

In async environments you'll want to take care not to issue these synchronous queries without ensuring that the query will indeed be executed against cache, since an error will be thrown otherwise.