

## Contents

- [Problem](#)
- [Solution](#)

This code sample shows how to query multiple models from a single *EntityManager*.

- **Platform:** Silverlight
- **Language:** C#
- **Download:** [Working with multiple models \(Silverlight\)](#)

## Problem

You have multiple Entity Models that represent different back-end datasources. How do you work with them using a single [EntityManager](#) so that they can share the same editing and transaction context?

## Solution

This solution demonstrates how to use an *EntityManager* to query from two separate models.

There are two models - *CoreModel* and *LeafModel*, which can be sourced from two different datasources. In this case, they are both from NorthwindIB, but the code does not know the difference.

By default, DevForce generates a typed *EntityManager* for each model and it contains properties for the basic queries for that model. However, any *EntityManager* can query for entities from any model. To do this, create a new [EntityQuery<T>](#) of the correct entity type, and then use the *.With(EntityManager mgr)* operator on the query to execute it:

```
var entityManager = new CoreModelEntities();
// Query for entities from the EntityManager's default model
entityManager.Customers.ExecuteAsync(op => {
    MessageBox.Show("Customers fetched: " + op.Results.Count().ToString());
});

// Query for entities from a different model
var query = new EntityQuery<Employee>();
query.With(entityManager).ExecuteAsync(op => {
    MessageBox.Show("Employees fetched: " + op.Results.Count().ToString());
});
```

```
Dim entityManager = New CoreModelEntities()
' Query for entities from the EntityManager's default model
entityManager.Customers.ExecuteAsync(Sub(op) _
    MessageBox.Show("Customers fetched: " & op.Results.Count().ToString()))
' Query for entities from a different model
Dim query = New EntityQuery(Of Employee)()
query.With(entityManager).ExecuteAsync(Sub(op) _
    MessageBox.Show("Employees fetched: " & op.Results.Count().ToString()))
```

You can also easily subclass and create your own typed *EntityManager* with whatever properties you need. The code generated *EntityManager* contains no "magic" and you can use it as a template for your own.

To navigate from one model to another, we suggest using a repository pattern in a third assembly to avoid circular references between the models. For example, you would call *Repository.GetSalesRep(someCustomer)* instead of *someCustomer.SalesRep*.