Contents

- Problem
- Solution
  - Next steps

You can easily access entities defined in multiple entity models from the same EntityManager. We'll show you how here.

- **Platform:** Silverlight
- **Language:** C#, VB
- **Download:** Working with Multiple Models

Problem

Applications often need multiple entity models. A sub-typed *EntityManager* is generated for each of those models, making it easier to query for entity types within that model. However, it is also easy to the same *EntityManager* with multiple models.

Solution

The code sample defines two entity models, each in its own assembly. The *Core* model holds a few entity types (in this case Customer, Product and Supplier), while the *Leaf* model holds the Employee, Order and OrderDetail entity types. In this sample, both models were defined using the NorthwindIB sample database, but in practice the models can be defined for different data sources.

You can easily imagine your own multi-model scenarios. Maybe you have entities that are used across all modules within your application, forming a core model, while you have module-specific entity models too, such as for sales or accounting.

Since this is a Silverlight application, we also defined corresponding Silverlight assemblies for each model, and linked the generated code in. If you add any additional business logic to entities which should be "shared" between the Silverlight client application and the EntityServer, be sure to also link those files.

The sub-typed *EntityManager* generated for each model contains *query* properties to make it easier to build EntityQueries. These query properties are only helpers, however, they don't limit what entity types the *EntityManager* can work with.

You can easily create your own *EntityQuery* when a helper query property isn't available, and the sample shows this. When working in the *CoreModelEntityManager*, the sub-typed *EntityManager* for the core model, it creates a query for an entity type in the leaf model:

```
var query = new EntityQuery<Employee>();
```

```
Dim query = New EntityQuery(Of Employee)()
```

Don't worry about the several *EntityQuery* constructor overloads, those are generally needed only for advanced uses or internally by the framework.

With the *EntityQuery* in hand, you can then execute the query against any *EntityManager*. In the sample, we show execution of queries for entities in both the core and leaf models.

Next steps

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

To navigate between entities in different models, we suggest using a repository pattern in a third assembly to avoid circular references among the models. For example, you might call Repository.GetSalesRep(someCustomer) instead of someCustomer.SalesRep. The additional layer of abstraction the repository provides isolates the modeling and data access mechanics, and improves testability.

When saving entities defined in multiple models from a single *EntityManager* the save will by default be transactional across the data sources. You may need to enable the Distributed Transaction Coordinator in this case.