Contents

In addition to database-backed classes - whether developed in the Database First, Model First or Code First style - DevForce also supports **Plain Old CLR Objects or POCOs**.

POCO classes in DevForce do not contain any DevForce-specific code: they don't inherit from *Entity* or implement *IEntity*, yet they can still be queried and saved using much of the standard DevForce pipeline.

Why POCO?

POCO is best suited for data that is not stored in a relational database: when you must access data from web services or XML files, for example, yet still want to treat these objects as "entities" within your application, and use the same query and save semantics.

If you do wish to use "POCO-style" classes backed by a database, check out our Code First support to see if it will meet your needs.

What can you do with a POCO object

- Query for entities with full DevForce LINQ support
- Save changes
- Use navigation properties
- Intercept queries and saves in your custom interceptors
- Secure objects, queries and saves
- Pass parameters to a server query method
- Project into a POCO type
- Use dynamic queries
- And more!

What can't you do

- POCO queries are not by default stored in the QueryCache. You can still execute *CacheOnly* queries, but the DevForce optimizations to use the query cache are not used.
- Query using property navigation. Navigation properties will not be lazily loaded, so you must retrieve the related entities when querying the root entity. For example, if both Order and OrderDetail are POCO types, anOrder.OrderDetails will not lazily load the OrderDetails for the order; you need to return both the order and its details from your Order query method.
- There is no built-in property change notification, change tracking, or verification. You can still implement INotifyPropertyChanged on your own, and manually perform property and instance level validation on POCO objects. To add and modify POCO objects you'll need to manually set their *EntityState*.
- Use property interceptors.

Overview

There are a few steps to implementing POCO classes, which we'll discuss in more depth in following topics. In brief:

- You'll write the code for your POCO classes. In an n-tier application there are a few rules to follow to ensure these objects can be transmitted across tiers.
- You'll write a "service provider" to provide query, and optionally save, support.
- For convenience, you'll probably want to add query methods to your *EntityManager* class so that you can refer to your POCO objects in LINQ queries in a manner similar to that which you use with other DevForce entities.

We cover each of these steps in the following topics.

Check out a code sample here to see POCO in action.