Contents

- A simple example
- POCO rules

A POCO class is a .NET class that has no dependencies on DevForce. In order for DevForce to be able to query and save instances of such types there are a few simple **implementation details** to consider.

A simple example

Let's start with a simple example. Below is a "State" class representing US states, with only two properties:

```
using System;
using System.ComponentModel.DataAnnotations;
namespace DomainModel {
public class State {
  [Key]
 public string Abbrev { get; set; }
 public string Name { get; set; }
 }
}
```

```
Imports System
Imports System.ComponentModel.DataAnnotations
Namespace DomainModel
 Public Class State
   <Key()>
  Public Property Abbrev() As String
  Public Property Name() As String
 End Class
End Namespace
```

You'll notice we added the Key attribute. This is optional, but indication of the primary key for an entity allows DevForce to cache instances of it, since each State object will have unique "identity" and can be easily differentiated from others. Since these objects are now cacheable, they can also be created, modified or deleted, and saved.

In the simple case here with the State class, since the US is unlikely to see any changes in its states, your application will therefore probably not be doing any additions, modifications or deletions, and you can omit the *Key* attribute. You would still be able to query for these objects, but the *EntityManager* would not add them to its cache, and you cannot execute *CacheOnly* queries for these objects.

We're not done yet, though, since you'll still have to tell DevForce how to query, and optionally save, your POCO objects, which we'll cover in a later topic. For now, we'll delve into some additional considerations.

POCO rules

It's important to note that if objects of a POCO class will only be queried and your application is not an n-tier or Silverlight application, then you probably don't need to do anything additional.

- POCO types without a *Key* attribute will not be added to the *EntityManager* cache and cannot make use of EntityAspect services.

The following rules apply to serialization. DevForce will serialize entities in a distributed application, when working with an *EntityCacheState*, and when saving entities.

- POCO types must be serializable:
  1) Mark the class and members with the *DataContract* and *DataMember* attributes, or
  2) The public properties of the class will be serializable if the class has a public parameterless constructor.
- POCO types must be "known" types. Indicating that a type is a known type allows it to be discovered by the DevForce infrastructure so that distributed queries and saves can be performed. There are several ways to identify the type as a known type to DevForce:
  (1) Decorate a property (or multiple properties) of the class with the ***Key*** attribute; or
  (2) Have the type implement the *IKnownType* interface; or
  (3) Implement the IKnownTypeProvider interface.

We saw above that the *Key* attribute is used to uniquely identify POCO objects, but it serves a double purpose: when DevForce sees this attribute it will automatically include the type in its known type processing.

- If any of your POCO class properties returns an object, an interface, or a base type, you'll need to indicate the possible concrete types via the KnownType attribute.
- POCO classes must be included in both the server- and client-side assemblies. Define them in a server-side project and link to them from a client-side project.