Contents

In order to provide query, and optionally save, capabilities for your POCO objects a **POCO Service Provider** is needed.

Overview

The service provider is simply a class deployed "server-side" and decorated with the *EnableClientAccess* attribute. DevForce uses this attribute to probe for POCO service providers. You can have any number of service providers: you can use a single provider for all your POCO types, or a provider per POCO types, or separate providers for query and save; it's up to you.

Your provider should contain "query methods", which we'll describe in more detail in the following topic. It might also contain "save by convention" methods. Saves of POCO objects can be accomplished in two ways: via convention-based methods or by an "adapter" implementation. Both will be described in a following topic.

DevForce will automatically call your query and save methods whenever an *EntityManager* query or SaveChanges method is called.

Example

The example below shows a simple POCO Service Provider which provides query and save logic for the POCO type "State" we saw earlier.

```
namespace DomainModel {
  [EnableClientAccess]
 public class USStatesServiceProvider {
   public IEnumerable<State> GetStates() { ... }
   public void InsertState(State entity) { ... }
   public void UpdateState(State current, State original) { ... }
   public void DeleteState(State entity) { ... }
  }
}
```

```
Namespace DomainModel
  <EnableClientAccess>
 Public Class USStatesServiceProvider
  Public Function GetStates() As IEnumerable(Of State)
...
  End Function
  Public Sub InsertState(ByVal entity As State)
...
  End Sub
  Public Sub UpdateState(ByVal current As State, ByVal original As State)
...
  End Sub
  Public Sub DeleteState(ByVal entity As State)
...
  End Sub
 End Class
End Namespace
```

All of the actual method code is stubbed out in the example so that the overall structure can be seen. Any of these methods can be omitted as long as the corresponding functionality is not needed. So if we never planned on inserting, deleting or updating a 'State' then the *InsertState*, *UpdateState* and *DeleteState* methods could all be omitted. Details of how such methods can be implemented are described later.

In the example, the method names *GetStates*, *InsertState*, *UpdateState*, *DeleteState* are all "convention" based. More explicit linkage between service methods and the underlying types can also be specified, as we'll see next.