Contents

DevForce provides the ability to **save POCO objects** by dispatching save calls to server-side *interceptor* methods. These methods come in two flavors: adapter-based or convention-based implementations. By default, DevForce will attempt to locate an adapter-based implementation; if unsuccessful, it will then look for a convention-based implementation.

Remember that you must define a primary key for any POCO class whose objects can be saved. Use the Key attribute to mark one or more properties as the primary key.

PocoSaveMode

The implementation mechanism can also be specified explicitly by setting the *PocoSaveMode* property of the *SaveOptions* instance passed into the *SaveChanges* call:

```
var so = new SaveOptions();
so.PocoSaveMode = PocoSaveMode.UseEntitySaveAdapter;
myEntityManager.SaveChanges(so);
```

```
Dim so = New SaveOptions()
so.PocoSaveMode = PocoSaveMode.UseEntitySaveAdapter
myEntityManager.SaveChanges(so)
```

The *PocoSaveMode* is an enumeration with the following values:

Value	Description
Default	Use an EntitySaveAdapter subclass if found, otherwise use save methods discovered via convention.
UseEntitySaveAdapter	Use an implementation of EntityServerPocoSaveAdapter.
UseMethodsDiscoveredViaConvention	Use methods discovered via convention.

Adapter-based saves

The adapter-based mechanism requires the existence of a server-side class that inherits from the *EntityServerPocoSaveAdapter*. If such a class is found, a single instance of this class will be created for each *SaveChanges* call, and the appropriate methods corresponding to each insert, update, or delete will be called for each entity to be saved. A single parameterless constructor is required.

The following bare-bones version of an EntitySaveAdapter implementation shows the methods you have available to override:

```
public class PocoSaveAdapter : EntityServerPocoSaveAdapter {
 public PocoSaveAdapter() {
  }
 public override void BeforeSave(IEnumerable entities, SaveOptions saveOptions) {
  }
 public override void AfterSave() {
  }

 public override void InsertEntity(object entity) {
  }

 public override void UpdateEntity(object entity, object originalEntity) {
  }

 public override void DeleteEntity(object entity) {
  }
}
```

```
Public Class PocoSaveAdapter
 Inherits EntityServerPocoSaveAdapter
 Public Sub New()
 End Sub
 Public Overrides Sub BeforeSave(ByVal entities As IEnumerable, ByVal saveOptions As SaveOptions)
 End Sub
 Public Overrides Sub AfterSave()
 End Sub
 Public Overrides Sub InsertEntity(ByVal entity As Object)
 End Sub
 Public Overrides Sub UpdateEntity(ByVal entity As Object, ByVal originalEntity As Object)
 End Sub
 Public Overrides Sub DeleteEntity(ByVal entity As Object)
 End Sub
End Class
```

Note that a single insert, update, or delete method handles saves for every entity type, so if save logic needs to be different for different types, the type of the entity passed in will need to be inspected and the execution branched appropriately.

The BeforeSave and AfterSave methods are called exactly once for each SaveChanges call and may be used for "setup" and "teardown" logic.

Convention-based saves

The convention-based mechanism uses a *POCO Service Provider*, which you've already implemented for your query support. The *POCO Service Provider* is a class decorated with the *EnableClientAccess* attribute. This class will include insert, update, and delete methods named according to the conventions defined below. It must also include a single parameterless constructor.

If such a class is found, a single instance of it will be created for each *SaveChanges* call, and the appropriate methods corresponding to each insert, update, or delete will be called for each entity to be saved.

For each type for which a persistence operation is provided, up to three methods must be written. The name of the method must begin with one of the prefixes defined below. The method must also conform to the signature defined below.

If you do not expect to save objects that require one or more of these signatures, then they do not have to be implemented. In other words, if you never plan to delete “Foo” type objects, then you do not need to implement the Delete method.

Insert Method

- *Prefixes*: Insert, Add, or Create
- *Signature*: void InsertX(T entity), where T is an entity Type

Update Method

- *Prefixes*: Update, Change, or Modify
- *Signature*: void UpdateX(T current, T original), where T is an entity Type

Delete Method

- *Prefixes*: Delete or Remove
- *Signature*: void DeleteX(T entity), where T is an entity Type

Below is an example implemention showing the use of the required conventional method names and signatures. *InsertFoo* could equally be named “AddFoo” or “CreateFoo”; UpdateFoo could be named “ChangeFoo” or “ModifyFoo”; and so forth.

```
[EnableClientAccess]
public class PocoSaveAdapterUsingConventions {
public PocoSaveAdapterUsingConventions() {
  }
public void InsertFoo(Foo entity) {
  // insert logic for any ‘Foo’ entities
}
public void InsertBar(Bar entity) {
  // insert logic for any ‘Bar’ entities
}
public void UpdateFoo(Foo current, Foo original) {
  // update logic for any “Foo’ entities
}
public void UpdateBar(Bar current, Bar original) {
  // update logic for any “Bar’ entities
}
public void DeleteFoo(Foo entity) {
  // Delete logic for any “Foo’ entities
}
public void DeleteBar(Bar entity) {
  // Delete logic for any “Bar’ entities
}
}
```

```
<EnableClientAccess()> _
```

```
Public Class PocoSaveAdapterUsingConventions
 Public Sub New()
 End Sub
 Public Sub InsertFoo(ByVal entity As Foo)
   ' insert logic for any ‘Foo’ entities
 End Sub
 Public Sub InsertBar(ByVal entity As Bar)
   ' insert logic for any ‘Bar’ entities
 End Sub
 Public Sub UpdateFoo(ByVal current As Foo, ByVal original As Foo)
   ' update logic for any "Foo’ entities
 End Sub
 Public Sub UpdateBar(ByVal current As Bar, ByVal original As Bar)
   ' update logic for any "Bar’ entities
 End Sub
 Public Sub DeleteFoo(ByVal entity As Foo)
   ' Delete logic for any "Foo’ entities
 End Sub
 Public Sub DeleteBar(ByVal entity As Bar)
   ' Delete logic for any "Bar’ entities
 End Sub
End Class
```