

Working with POCO Entities

Article 08/29/2011

The Entity Framework enables you to use custom data classes together with your data model without making any modifications to the data classes themselves. This means that you can use "plain-old" CLR objects (POCO), such as existing domain objects, with your data model. These POCO data classes (also known as persistence-ignorant objects), which are mapped to entities that are defined in a data model, support most of the same query, insert, update, and delete behaviors as entity types that are generated by the Entity Data Model tools.

Mapping Requirements

To use POCO entities with a data model, the name of the entity type must be the same as the custom data class, and each property of the entity type must map to a public property of the custom data class. The names of the types and each of the mapped properties must be equivalent. For information on how to modify entities in a conceptual model, see [How to: Create and Modify Entity Types](#).

ⓘ Note

Mapping POCO entities is not supported if any mapping attributes are applied to custom data classes, including [EdmSchemaAttribute](#) at the assembly level.

You can use the POCO template to generate persistence-ignorant entity types from a conceptual model. The template is not included with Visual Studio but can be downloaded from the [visual studio gallery](#).

Proxy Object Creation

If you want the Entity Framework to track changes in your POCO classes as the changes occur and support lazy loading of the related objects, your POCO classes must meet the requirements described in the [Requirements for Creating POCO Proxies](#) topic.

When proxy object creation is enabled for POCO entities, changes that are made to the graph and the property values of objects are tracked automatically by the Entity Framework

as they occur. For information about change tracking options with and without proxies, see [Tracking Changes in POCO Entities](#).

You can have a mix of POCO entities and proxy entity objects. To disable creating proxy objects, set the value of the [ProxyCreationEnabled](#) property to **false** on the instance of [ObjectContextOptions](#) that is returned by the [ContextOptions](#) property on the [ObjectContext](#) :

VB

```
' Disable proxy object creation.  
context.ContextOptions.ProxyCreationEnabled = False
```

For more information, see [How to: Create a POCO Entity with Proxies](#).

Serializing POCO Proxies

Windows Communication Foundation (WCF) cannot directly serialize or deserialize proxies because the [DataContractSerializer](#) can only serialize and deserialize known types, and proxy types are not known types. When you need to serialize POCO entities, disable proxy creation or use the [ProxyDataContractResolver](#) class to serialize proxy objects as the original POCO entities. To disable proxy creation, set the **ProxyCreationEnabled** property to **false**.

The **ProxyDataContractResolver** class maps proxy types to POCO types during serialization. To instruct the **DataContractSerializer** to use the **ProxyDataContractResolver** class in service operations, define an attribute class (that will be applied to the service operations) that internally uses the **ProxyDataContractResolver** to map proxy types to pure POCO types. Associate this attribute class with methods that are part of a service contract in your WCF application.

The client will receive and deserialize the actual POCO entities. These classes will not have the lazy-loading and change-tracking capabilities of proxy objects. To track changes in these entities across tiers, use self-tracking entities. Self-tracking entities are POCO entities that do not have a dependency on the Entity Framework and contain their own change-tracking logic. For more information, see [Walkthrough: Serialize Self-Tracking Entities](#).

For more information, see [Walkthrough: Serialize POCO Proxies with WCF](#).

Proxy objects can be serialized and deserialized with binary serialization. However, when serializing an object graph across [AppDomain](#) boundaries, you must ensure that the proxy type definitions exist in the target environment. If the types do not exist, deserialization will fail. To make sure that the type exists, use [CreateProxyTypes](#) .

ⓘ **Note**

Even though an object of a proxy type is created, it does not have the lazy-loading and change-tracking capabilities of a proxy.

With binary serialization and data contract serialization, related objects are serialized together with the primary object. Lazy loading performs a query for each relationship navigation property that is accessed, and both binary and WCF data contract serializers access all relationship navigation properties. This can cause many unexpected queries to be performed during serialization. If you do not disable proxy type generation (which would disable lazy loading), explicitly disable lazy loading, as in the following example.


VB

```
' Disable lazy loading.
context.ContextOptions.LazyLoadingEnabled = False
```

For more information, see [Serializing Objects](#).

Summary of Proxy Specific APIs

The following APIs are relevant to working with POCO proxies:

 Expand table

Member	Description
CreateObject	Creates a new POCO proxy object, if your POCO class meets the requirements described in the Requirements for Creating POCO Proxies topic and the ProxyCreationEnabled is set to true , otherwise creates an object of generic argument type. The generic argument of this method can be any concrete reference type (abstract classes, interfaces and value types are not supported). If the generic type is a CLR type that is not map to the conceptual model, or a type that does not meet the requirements for

Member	Description
	<p>proxy creation, the method will attempt to use any parameterless constructor of the passed type to create and return a new instance of the type. In addition to the CreateObject on ObjectContext you can use CreateObject and CreateObject methods on ObjectSet . For more information, see How to: Create a POCO Entity with Proxies.</p> <p>This method does not add the created object to the object context. To add the object to the context you need to use methods described in the Creating, Adding, Modifying, and Deleting Objects topic.</p>
ProxyCreationEnabled	With this flag set to true the Entity Framework will attempt to create proxies for your POCO entities. ProxyCreationEnabled flag is set to true by default.
GetObjectType	The System.Data.Objects.ObjectContext.GetObjectType(System.Type) is a static method that returns the POCO type that the specified proxy derives from if a proxy type was passed as an argument. If a non-proxy type was passed, the same type is returned by this method.
CreateProxyTypes	<p>Creates a set of proxy types for the specified POCO types according to the metadata loaded in the ObjectContext. This method does not instantiate objects of created proxy types. For example, the following code will create proxies for the <code>Customer</code> and <code>Order</code> POCO classes:</p> <pre>context.CreateProxyTypes(new Type[] { typeof(Customer), typeof(Order) });</pre> <p>When you load objects from the data source or created new objects with CreateObject, the Entity Framework creates proxy types for your POCO classes during runtime. However, there could be scenarios when you might want to create the proxy types in advance. For example, when serializing an object graph across AppDomain boundaries you might want to make sure that the proxy types actually exist in the target environment. If the types do not exist, the deserialization will fail.</p>
GetKnownProxyTypes	The GetKnownProxyTypes is a static method that returns an enumeration containing all the proxy types that have been created so far in the AppDomain . In serialization scenarios, the method can be used to obtain all the types that the target environment should already contain and the serializer should recognize.

In This Section

[Requirements for Creating POCO Proxies](#)

[Loading Related POCO Entities](#)

[Tracking Changes in POCO Entities](#)

[How to: Define POCO Entities](#)

[How to: Define a Custom Object Context](#)

[How to: Customize Modeling and Mapping Files to Work with Custom Objects](#)

[How to: Define a Custom Object Context](#)

[How to: Create a POCO Entity with Proxies](#)

[How to: Explicitly Load POCO Entities](#)

[How to: Detect Changes in POCO Entities](#)

[How to: Change Relationships Between POCO Entities](#)

Additional Resources

The following is the series of blog posts on POCO from [ADO.NET team blog](#) .

[POCO in the Entity Framework 4 - Part 1](#)

[POCO in the Entity Framework 4 - Part 2](#)

[POCO in the Entity Framework 4 - Part 3](#)

See Also

Concepts

[Customizing Objects](#)

[Defining and Managing Relationships](#)