Contents

- The Entity Data Model (EDM)
- Entity Framework and the EDM
- Entity Framework Code First
- Alternatives to Entity Framework

Most DevForce developers describe entities in relation to a database schema using an Entity Framework **Entity Data Model (EDM)**

That EDM consists of three parts: a definition of conceptual entities, a definition of database objects, and the mapping between them. The EDM is represented in an XML file called the EDMX. You use the Entity Framework's EDM Designer to view and modify that EDMX rather than edit the XML file itself. This topic covers these points.

The Entity Data Model (EDM)

You have some kind of "entity data model" in mind if you have entities that are supported by a database. You've got entity classes, database objects and a mapping between them. That's the essence of an **EDM**.

The EDM could be implicit in the code you write. For example, you can write a data access layer (DAL) specifically for your application such that it pushes and pulls data, record-by-record, field-by-field, into and out of each entity type. This DAL has intimate and detailed knowledge of both the entities and the database. The EDM is hiding in there somewhere, buried in the details of each operation.

Writing an application this way is hard work; maintaining it will be almost as hard. We would recommend this approach only for high performance systems with few entity types. It has its place and you can write a DevForce application that relies in whole or in part on a DAL of this kind; look in the documentation for a discussion of POCO (Plain Old CLR Objects) entities.

Most developers prefer to separate the EDM from the mechanics of data access. They expect the data access machinery to read the EDM and follow instructions expressed in terms of that EDM.

There are three popular ways to produce such an EDM.

1. Derive it by examining an existing database – the "Data First" approach.
2. Write a purely conceptual entity model and generate an EDM from that – the "Model First" approach.
3. Examine the entity classes themselves and derive an EDM from those classes – the "Code First" approach.

Readers familiar with the Microsoft Entity Framework (EF) will recognize these as the three approaches to mapping supported by the latest version of Entity Framework.

Entity Framework and the EDM

DevForce is works well with the Entity Framework in all three modes. The Database First and Model First approaches are popular with developers who are working with a large, pre-existing database. In these modes, you use the Entity Framework EDM Designer in Visual Studio to build an Entity Data Model (EDM); that model is represented as an XML file (the EDMX). You can edit the EDMX with a text editor but you are more likely to use the EDM Designer to maintain it.

Every time you change the model, DevForce reads the revised EDMX file and re-generates your entity class files. You compile these class files - supplemented by your custom business logic - into an entity class model and deploy that model both on the client and the Application Server.

You also deploy the EDMX file to the Application Server. The Entity Framework, acting as a data access layer (DAL) guided by the EDMX, turns queries and other commands into database operations and moves raw database data into and out of the properties of actual entity instances.

Entity Framework Code First

Code First is a style of Entity Framework model development in which you write the entity classes and map them to the database by hand. You do not use the EDM Designer and there is no EDMX file. The Entity Framework constructs an internal EDM dynamically by examining the entity classes themselves (we're simplifying). It uses this internal EDM, as it would the EDMX, to guide its persistence operations.

Learn how to develop Code First with DevForce.

Alternatives to Entity Framework

You can back your entity classes with other ORM tools such as NHibernate. You can also write a model backed by a non-relational data sources. DevForce doesn't provide native support for these alternatives; you may wish to consider using the DevForce POCO development approach when building models with these technologies.