

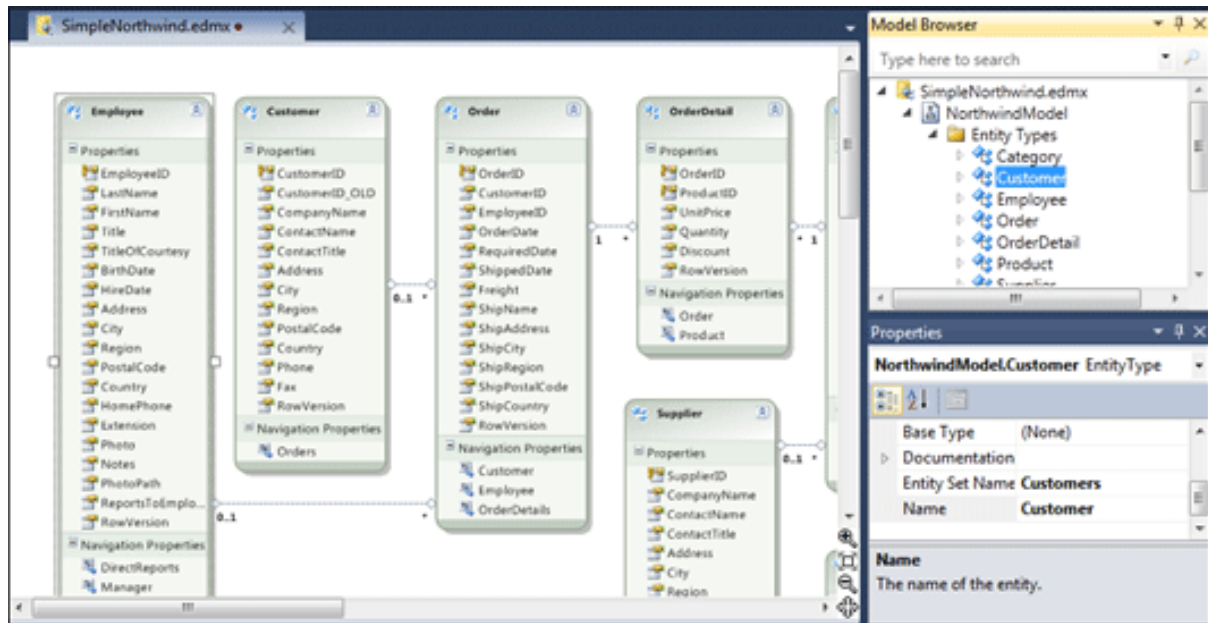
## Contents

- [DevForce EDM Designer extension](#)
- [EDM Designer modeling approaches](#)
- [Code generation](#)
- [Iterative model development](#)

You develop your [Entity Data Model \(EDM\)](#) in Visual Studio using a graphical design tool called the **EDM Designer**

Details of the **EDM Designer** are beyond the scope of this overview. Indeed, they are beyond the scope of DevForce documentation although we do guide you through the essentials of the design experience in the development section on [modeling](#).

For now we must be satisfied with a screen shot and a high altitude discussion.



Here we see the EDM designer with three open panels:

1. The design canvas displaying a diagram of the conceptual entities and their associations.
2. The **Model Browser**, a searchable tree-view of the model's entity types, their properties, and their associations.
3. The **Properties** panel showing an editable property grid of the currently selected EDM node. In this example we see the properties for the Customer entity selected in the Model Browser.

## DevForce EDM Designer extension

The Entity Framework EDM Designer only collects the information it needs for [Entity Framework](#)'s native code generator. DevForce needs more information in order to generate its entity classes. The [DevForce EDM Designer extension](#), which was installed with DevForce, adds features to the base EDM Designer that help the developer provide that extra information.

## EDM Designer modeling approaches

The EDM in our example could have evolved in one of two ways:

1. "Data First" by pointing to a pre-existing database and asking the tool to generate the EDM from the database schema.
2. "Model First" by manually drawing conceptual entities and their associations on the design canvas, configuring their attributes in the "Properties" panel, and (optionally) generating the Data Definition Language (DDL) script to create the database from this conceptual entity model.

"Data First" works well when you have an existing database. It's also popular with those who like to build the database slightly ahead of the model. "Data First" is the most common approach by far.

"Model First" works well when you are starting fresh and prefer to explore entity model possibilities unencumbered by a database schema. You don't have to generate the corresponding database definition until you are ready.

With DevForce you can delay building a database for months by developing entirely offline, living off an [entity cache](#) populated with development data.

With Data First you always get a complete EDMX. With Model First, until you generate the database definition script, the EDMX has a Conceptual Model definition (CSDL) but no Storage Model definition (SSDL) and no mapping definition (C-S).

The Entity Framework [data access layer \(DAL\)](#) won't function without an SSDL and C-S. You won't be putting this application into production.

Joe Fancey's MSDN article, [Model-First in the Entity Framework 4](#), is an excellent introduction to the EDMX and both approaches to model building in EF.

## Code generation

The

You can generate the .NET entity classes with the CSDL alone. Saving changes to the EDMX file causes a code generator to read the CSDL and emit .NET class files; the CSDL contains all the information it needs for this purpose.

Entity Framework has its own code generator (several actually). When DevForce tooling is installed and enabled, the DevForce code generator takes over and emits DevForce entity class files.

This overview introduces you to DevForce integration with Entity Framework and [code generation](#), a topic covered extensively in the development section on [modeling](#).

## Iterative model development

Your Entity Model will change during the life of the application. You will add and modify database tables, views, columns, and foreign key relationships. The EDM designer can re-read the database schema and update all parts of the EDM to reflect these changes.

The EDM designer helps you adjust the Conceptual Entity Model manually to keep up with the Storage Model changes or to refine your thinking about how the database structures should be represented as entities and associations.

The .NET entity class files are regenerated automatically when you save your changes.