Entity Framework Overview

The Entity Framework is a set of technologies in ADO.NET that support the development of data-oriented software applications. Architects and developers of data-oriented applications have struggled with the need to achieve two very different objectives. They must model the entities, relationships, and logic of the business problems they are solving, and they must also work with the data engines used to store and retrieve the data. The data may span multiple storage systems, each with its own protocols; even applications that work with a single storage system must balance the requirements of the storage system against the requirements of writing efficient and maintainable application code.

The Entity Framework enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to concern themselves with the underlying database tables and columns where this data is stored. With the Entity Framework, developers can work at a higher level of abstraction when they deal with data, and can create and maintain data-oriented applications with less code than in traditional applications. Because the Entity Framework is a component of the .NET Framework, Entity Framework applications can run on any computer on which the .NET Framework starting with version 3.5 SP1 is installed.

The following sections in this topic provide more detail about the Entity Framework:

Giving Life to Models

A longstanding and common design approach when building an application or service is the division of the application or service into three parts: a domain model, a logical model, and a physical model. The domain model defines the entities and relationships in the system that is being modeled. The logical model for a relational database normalizes the entities and relationships into tables with foreign key constraints. The physical model addresses the capabilities of a particular data engine by specifying storage details such as partitioning and indexing.

The physical model is refined by database administrators to improve performance, but programmers writing application code primarily confine themselves to working with the logical model by writing SQL queries and calling stored procedures. Domain models are generally used as a tool for capturing and communicating the requirements of an application, frequently as inert diagrams that are viewed and discussed in the early stages of a project and then abandoned. Many development teams skip creating a conceptual model and begin by specifying tables, columns, and keys in a relational database.

The Entity Framework gives life to models by enabling developers to query entities and relationships in the domain model (called a *conceptual* model in the Entity Framework) while relying on the Entity Framework to translate those operations to data source-specific commands. This frees applications from hard-coded dependencies on a particular data source. The conceptual model, the storage model, and the mappings between the two are expressed in XML-based schemas and defined in files that have corresponding name extensions:

- Conceptual schema definition language (CSDL) defines the conceptual model. CSDL is the Entity Framework's implementation of the Entity Data Model. The file extension is .csdl.
- Store schema definition language (SSDL) defines the storage model, which is also called the logical model. The file extension is .ssdl.
- Mapping specification language (MSL) defines the mappings between the storage and conceptual models. The file extension is .msl.

The storage model and mappings can change as needed without requiring changes to the conceptual model, data classes, or application code. Because storage models are provider-specific, you can work with a consistent conceptual model across various data sources.

The Entity Framework uses these model and mapping files to create, read, update, and delete operations against entities and relationships in the conceptual model to equivalent operations in the data source. The Entity Framework even supports mapping entities in the conceptual model to stored procedures in the data source. For more information, see CSDL, SSDL, and MSL Specifications.

Mapping Objects to Data

Object-oriented programming poses a challenge for interacting with data storage systems. Although the organization of classes frequently mirrors the organization of relational database tables, the fit is not perfect. Multiple normalized tables frequently correspond to a single class, and relationships between classes are often represented differently than relationships between tables are represented. For example, to represent the customer for a sales order, an **Order** class might use a property that contains a reference to an instance of a **Customer** class, while an **Order** table row in a database contains a foreign key column (or set of columns) with a value that corresponds to a primary key value in the **Customer** table. A **Customer** class might have a property named **Orders** that contains a collection of instances of the **Order** class, while the **Customer** table in a database has no comparable column. The Entity Framework provides developers with the flexibility to represent relationships in this way, or to more closely model relationships as they are represented in the database. For more information, see Defining and Managing Relationships.

Existing solutions have tried to bridge this gap, which is frequently called an "impedance mismatch", by only mapping object-oriented classes and properties to relational tables and columns. Instead of taking this traditional approach, the Entity Framework maps relational tables, columns, and foreign key constraints in logical models to entities and relationships in conceptual models. This enables greater flexibility both in defining objects and optimizing the logical model. The Entity Data Model tools generate extensible data classes based on the conceptual model.

These classes are partial classes that can be extended with additional members that the developer adds. By default, the classes that are generated for a particular conceptual model derive from base classes that provide services for materializing entities as objects and for tracking and saving changes. Developers can use these classes to work with the entities and relationships as objects related by associations. Developers can also customize the classes that are generated for a conceptual model. For more information, see Working with Objects and How to: Customize Object-Layer Code Generation.

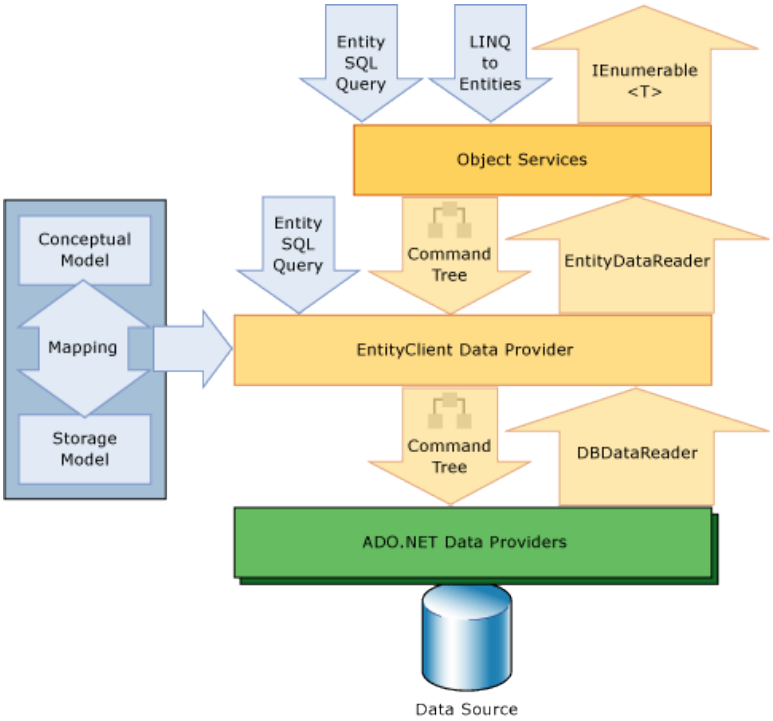
Accessing and Changing Entity Data

More than just another object-relational mapping solution, the Entity Framework is fundamentally about enabling applications to access and change data that is represented as entities and relationships in the conceptual model. The Entity Framework uses information in the model and mapping files to translate object queries against entity types represented in the conceptual model into data source-specific queries. Query results are materialized into objects that the Entity Framework manages. The Entity Framework provides the following ways to query a conceptual model and return objects:

- LINQ to Entities. Provides Language-Integrated Query (LINQ) support for querying entity types that are defined in a conceptual model. For more information, see LINQ to Entities.
- Entity SQL. A storage-independent dialect of SQL that works directly with entities in the conceptual model and that supports Entity Data Model concepts. Entity SQL is used both with object queries and queries that are executed by using the EntityClient provider. For more information, see Entity SQL Overview.
- Query builder methods. These methods enable you to construct Entity SQL queries using LINQ-style query methods. For more information, see Query Builder Methods.

The Entity Framework includes the EntityClient data provider. This provider manages connections, translates entity queries into data source-specific queries, and returns a data reader that the Entity Framework uses to materialize entity data into objects. When object materialization is not required, the EntityClient provider can also be used like a standard ADO.NET data provider by enabling applications to execute Entity SQL queries and consume the returned read-only data reader. For more information, see EntityClient Provider for the Entity Framework.

The following diagram illustrates the Entity Framework architecture for accessing data:



The Entity Data Model Tools can generate a class derived from ObjectContext that represents the entity container in the conceptual model. This object context provides the facilities for tracking changes and managing identities, concurrency, and relationships. This class also exposes a **SaveChanges** method that writes inserts, updates, and deletes to the data source. Like queries, these changes are either made by commands automatically generated by the system or by stored procedures that are specified by the developer. For more information, see Creating, Adding, Modifying, and Deleting Objects.

Data Providers

The **EntityClient** provider extends the ADO.NET provider model by accessing data in terms of conceptual entities and relationships. It executes queries that use Entity SQL. Entity SQL provides the underlying query language that enables **EntityClient** to communicate with the database. For more information, see EntityClient Provider for the Entity Framework.

The Entity Framework includes an updated SqlClient Data Provider that supports canonical command trees. For more information, see SqlClient for the Entity Framework.

Entity Data Model Tools

Together with the Entity Framework runtime, the .NET Framework version 4 includes the EDM Generator (EdmGen.exe). This command prompt utility connects to a data source and generates model and mapping files based on a one-to-one mapping between entities and tables. It also uses a conceptual model file (.csdl) to generate an object layer file that contains classes that represent entity types and the **ObjectContext**. For

more information, see EDM Generator (EdmGen.exe).

Visual Studio 2010 includes rich tool support for generating and maintaining model and mapping files in a Visual Studio application. The Entity Data Model Designer supports creating advanced mapping scenarios, such as table-per-type and table-per-hierarchy inheritance and split entities that map to multiple tables. For more information, see ADO.NET Entity Data Model Designer.

Learning More

The following topics enable you to learn more about the Entity Framework:

Getting Started (Entity Framework)
Provide information about how to get up and running quickly using the Quickstart, which shows how to create a simple Entity Framework application.

Quickstart
Shows you how to use the Entity Data Model tools with to quickly create your first Entity Framework application.

Entity Framework Terminology
Defines many of the terms that are introduced by the Entity Data Model and the Entity Framework and that are used in Entity Framework documentation.

Entity Framework Resources
Provides links to conceptual topics and links to external topics and resources for building Entity Framework applications.

See Also

Concepts

ADO.NET Entity Framework