Contents

This topic explains the basics of connecting to a database with DevForce Code First.

Default connections

In Code First, SQL Server Express is the default *connection factory*. If you make no accomodations - such as defining a connection string in the app.config, or modifying the DefaultConnectionFactory or its BaseConnectionString - then a SQL Express database will be used. The database name will be that of the *DataSourceKeyName*, and the database will be created if it doesn't exist.

This is generally desirable behavior, especially when beginning development and you are not tied to a legacy database. But what if you don't have SQL Express installed, or want to use an existing database, possibly from another database provider?

You still can. You can reset the *DefaultConnectionFactory* or you can add a connectionString entry in a configuration file, both of which are described in detail in the living without SQL Express topic.

In the following we describe some of the basics of connection strings and data source key names in Code First.

The Code First database connection string

Entity Framework Code First uses standard ADO.NET connection strings, not EDM connection strings.

The typical place for connection string is in the <*connectionStrings*> section of a configuration file (*app.config* or *web.config*). For example, here's a Code First connection string:

```
<add
   name="ProductEntities"
   connectionString="data source=.;initial catalog=CodeFirstDemoDb;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework"
   providerName="System.Data.SqlClient"
/>
```

DataSourceKeyName and the name of the connection string

DevForce and Entity Framework prefer to acquire connection strings by name.

Putting a raw connection string in your code is widely considered a bad practice and strongly discouraged by DevForce.

The connection string name in the example above is "ProductEntities".

The connection string name is important because DevForce correlates that name with your entity model's ***DataSourceKeyName***. Every DevForce entity type has a *DataSourceKeyName*. So to query and save entity data you may imagine that DevForce does something like this: look at the entity type, find its *DataSourceKeyName*, locate the database connection string that goes with that key name, and connect to the database (via Entity Framework) using that connection string.

The entity types of an entity model all share the same *DataSourceKeyName*. They get their key name in one of two ways:

1. From your custom *EntityManager* class if you have not written a custom *DbContext*
2. From your custom *DbContext* class if you wrote one.

Option #1: *EntityManager* only

If you wrote an *EntityManager* class and did not write a *DbContext* class, the *DataSourceKeyName* comes from your *EntityManager* class.

The *DataSourceKeyName* will be the name of your *EntityManager* class ("ProductEntities") if you declare it like this:

```
// OK PRACTICE
public class ProductEntities : EntityManager {...}
```

In this example, the name of the class ("ProductEntities") is the same as the connection string name defined above, so DevForce (and Entity Framework) will connect to the proper database at runtime.

Suppose you want to change the name of your *EntityManager* but keep the same connection string name. You can specify the *DataSourceKeyName* explicitly by applying a *DataSourceKeyName* attribute to the *EntityManager* class:

```
// BETTER PRACTICE: using the DataSourceKeyName attribute
[DataSourceKeyName("ProductEntities")]
public class ProductManager : EntityManager {...}
```

The *EntityManager* is called "Product*Manager*" but the *DataSourceKeyName* remains "Product*Entities*" because you named it that with the *DataSourceKeyName* attribute.

Do not put an underscore ("_") character in your *DataSourceKeyName*. The underscore has a special meaning in DevForce; it is used as the distinguishing character in a "data source key extension".

Option #2: *DbContext*

When you write a custom *DbContext*, the *DbContext* trumps the *EntityManager* in determining the *DataSourceKeyName*. We can ignore the *EntityManager* ... whether or not you wrote one.

The *DataSourceKeyName* now will be the name of your *DbContext* class ("ProductDbContext") if you declare it like this:

```
// NOT GOOD PRACTICE: Missing the DataSourceKeyName attribute
public class ProductDbContext : DbContext
 {
  // Constructor with connection parameter
  public ProductDbContext(string connection) : base(connection) {...}
 }
```

The name of the *DbContext* class ("ProductDbContext") is not the same as the connection string name ("ProductEntities") so DevForce **will not find the proper connection string!!!**

You could change the connection string name to "ProductDbContext" but that doesn't seem right. We think that *DbContext* class names make **poor connection string names** so we strongly recommend that you **always add the *DataSourceKeyName* attribute** as shown here:

```
// GOOD PRACTICE: using the DataSourceKeyName attribute
[DataSourceKeyName("ProductEntities")]
public class ProductDbContext : DbContext
 {
  // Constructor with connection parameter
  public ProductDbContext(string connection) : base(connection) {...}
 }
```

The *DbContext* is still called "ProductDbContext" but the *DataSourceKeyName* remains "ProductEntities" and DevForce will be able to find the corresponding connection string in the configuration file.

Important: Notice that the *ProductDbContext* has a **constructor that takes a string parameter**. DevForce uses this constructor to pass the proper connection string to Entity Framework.

Always add this constructor to your *DbContext*.

In fact, DevForce will throw a build error if you apply the *DataSourceKeyName* attribute and the *DbContext* lacks this constructor.

A missed connection

What if ...

- the project doesn't have a configuration file? ... OR
- there is no connection string in the configuration file with a name that matches the *DataSourceKeyName*?

In these cases, the Entity Framework tries to create a database for you in **SQL Server Express**. The name of the database will be the same as the *DataSourceKeyName* and its schema will conform to the entity model (with the mappings you defined for it).

This may be exactly what you want. Many developers use Code First to create a working database in the early phases of application development when the ultimate schema and data don't really matter.

Just don't be surprised.

Build vs. Runtime

When you read about generating metadata, you learned that a database connection is made to obtain storage model metadata when DevForce generates the *.ibmmx* metadata file. To avoid build time errors, you may want to add an app.config file to the model project to provide database information.

If you have a separate model project, remember that the app.config in the model project will not be used at runtime. At runtime, the app.config in the application executable project, or the web.config, will be used when searching for database connection strings.

Learn More

Your application may have unusual needs. Maybe you have more than one model or more than one database. Maybe you want to determine the connection string dynamically based on runtime characteristics of the client application. Maybe you are converting a "Database First" model to "Code First" ... in which case you'll need to fix the connection string(s).

These and other advanced scenarios are covered in the "Advanced database connections" topic.