Contents

You can use SQL Server Compact 4.0 with DevForce Code First; we'll describe how here.

Installation matters

There are several different ways in which SQL CE can be installed, and these differences will impact how you configure your application, both at build and run time.

Using Nuget package EntityFramework.SqlServerCompact

If you installed the package to a web project it will modify the web.config to add an entry to *DbProviderFactories*, but if you installed the package to another type of project you will need to manually add the following entry to the project's configuration file:

```
<system.data>
  <DbProviderFactories>
    <remove invariant="System.Data.SqlServerCe.4.0" />
    <add name="Microsoft SQL Server Compact Data Provider 4.0"
         invariant="System.Data.SqlServerCe.4.0"
         description=".NET Framework Data Provider for Microsoft SQL Server Compact"
         type="System.Data.SqlServerCe.SqlCeProviderFactory, System.Data.SqlServerCe, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=89845dcd8080cc91" />
  </DbProviderFactories>
</system.data>
```

For build time support, this entry must be found in the configuration file for your model project. For run time support, this entry must be in the server's configuration file (or the application's configuration file if a 2-tier application). See here for more information on build vs. run time configuration requirements when using DevForce Code First.

The package should also add references to *System.Data.SqlServerCe* and *System.Data.SqlServerCe.Entity*; if not, add references to these assemblies to your project.

Full install

If you instead installed SQL CE from the Microsoft download site, the *DbProviderFactories* entry will have been added to your machine.config.

You will still need to add references to *System.Data.SqlServerCe* and *System.Data.SqlServerCe.Entity* in your project.

Set the DefaultConnectionFactory

To tell DevForce that you're using a non-standard provider you'll also need to set the *DefaultConnectionFactory*.

```
static ProductDbContext() {
  Database.DefaultConnectionFactory = new SqlCeConnectionFactory("System.Data.SqlServerCe.4.0");
}
```

In order to find the *DefaultConnectionFactory* setting at both build and run time, the best place to make this change is in the static constructor for your custom *DbContext*.

If you wish to set a connection string, you can do so in either the *connectionStrings* section of the configuration file, or via the *baseConnectionString* argument to the constructor above.

(The need for the *DefaultConnectionFactory* setting here is a DevForce requirement which may be removed in a later release.)

Don't use TransactionScope

The following applies only to versions of DevForce prior to 7.2.0.

DevForce wraps all queries and saves in a TransactionScope, but this can cause escalation to a distributed transaction when using SQL CE with Code First. Since SQL CE does not support distributed transactions you'll need to disable the *TransactionScope*. You do this via the *TransactionSettings.Default* static property. Here's an example - note that it's the **false**

argument to the constructor which disables *TransactionScope*, the other arguments determine the transaction isolation level and timeout, with defaults shown here:

```
TransactionSettings.Default = new TransactionSettings(System.Transactions.IsolationLevel.ReadCommitted, TimeSpan.FromSeconds(30),
false);
```

Note that your saves will still be transactional, as a database transaction is still used. The change ensures that the database transaction will not be enlisted in a *TransactionScope*.

You need set the *TransactionSettings.Default* once only.