Contents

- Data source keys
  - In the EDM Designer
  - At run time
- Data source extensions
  - EdmKeys vs. ConnectionStrings
- Dynamic connection strings
- Anatomy of a connection string
- Security and connection strings

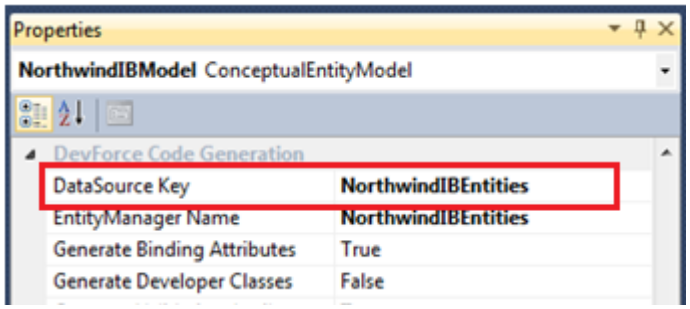
You have several options on how your application will **connect to a data source**. We'll discuss them here.

Data source keys

DevForce uses the concept of a *Data Source Key* to identify a data source. The key is used both at design-time, when the name of the *DataSourceKey* is tied to the generated code for an entity model, and again at run time when the *EntityManager* will query and save entities from the entity model.

In the EDM Designer

When you created your entity model in the Visual Studio EDM Designer, you specified a *DataSourceKey* for the model.



The *EntityManager Name* and *DataSource Key* name do not have to be the same. They often are in samples, since they both default to the *Entity Container Name* supplied in the EDM Designer.

This key name is also written into the generated code, to associate the entities in a model with this schema name:

```
[IbEm.DataSourceKeyName(@"NorthwindIBEntities")]
public partial class Employee : IbEm.Entity { .. }
```

```
<IbEm.DataSourceKeyName("NorthwindIBEntities")>
Partial Public Class Employee
 Inherits IbEm.Entity
End Class
```

By default, the *DataSourceKey* is given the same name as the *Entity Container Name*, and thus points to the connection string the designer adds to the config file in your model project:

```
<connectionStrings>
  <add name="NorthwindIBEntities" connectionString="…" />
</connectionStrings>
```

At run time

The EDM Designer created the connection string from a database used at design time. In the early phases of development, you'll likely use this same database, and connection string, for your development and testing. As you query and save entities, DevForce uses the *DataSourceKeyName* to help find the connection information to the run time database.

The connection information must be in the configuration file used by the *EntityServer*. If your entity model was generated into a class library (which is recommended), the app.config the EDM generated for design-time support won't ever be found or used at run time. So, your first task is ensuring that you copy or add the connection information to the run time config file. In an n-tier application, this config file will be on the server, since an n-tier client application does not directly work with the data tier.

See the topic on config file discovery for more information on how the configuration file is found at run time.

We're not done yet, though. DevForce uses two pieces of information to find the appropriate run time connection information: the ***data source key name*** combined with the ***data source extension***, which we'll discuss next.

Data source extensions

You may not have realized it, but when you constructed your EntityManager you also supplied it with a *data source extension*. The default *extension* is an empty string, meaning of course that no extension is used. An *extension* allows an *EntityManager* to target a specific run time database: for example it might be instructed to use "Dev" or "Test", or the *extension* might indicate that the *EntityManager* should use the databases for a particular tenant in a multi-tenant application.

DevForce uses the *DataSourceKeyName* to identify the data source schema, combined with the *data source extension* to indicate the specific schema-matching database to use. It's these two pieces of information which DevForce uses when searching for the "connection information" in your configuration file.

For example, if we construct an EntityManager with the "Test" *data source extension*:

```
manager = new NorthwindEntities(dataSourceExtension: "Test");
```

```
manager = New NorthwindEntities(dataSourceExtension:= "Test")
```

DevForce will use the *data source key*, in this case "NorthwindIBEntities", combined with the *data source extension* of "Test" to search the configuration for the best match. But what specifically is it searching for? We cover that next.

EdmKeys vs. ConnectionStrings

Above we used the vague term "connection information". We did this intentionally, since there are two ways you can provide connection information to DevForce in a configuration file:

- The first is with *connectionStrings*. These are the .NET standard way of defining database connection information in a configuration file. With a *connectionString* element, you supply a name and a connectionString containing the appropriate Entity Framework connection information.
- The second method is using a DevForce EdmKey. An *edmKey* contains connection information, as a *connectionString* will, but also additional DevForce attributes, such as *logTraceString*, which writes the EF-generated SQL query to the debug log.

In the sample above, DevForce will search for an *edmKey* or *connectionString* named "NorthwindIBEntities_Test". The underscore is the separator character DevForce uses, and allows you to construct multi-level extensions too. Remember not to use the underscore in the *data source extension* you pass into the *EntityManager* however.

DevForce will search for the best match. If you have both an *edmKey* and *connectionString* with the same name, DevForce will give preference to the *edmKey*. If DevForce can't find an exact match, it will drop the last extension from the string and then continue the search, and so on for all extensions provided. Note that the search is not case sensitive. If a match isn't found, you'll receive an error when attempting to query or save.

Unless providing connection information dynamically, you must ensure that either an *edmKey* or *connectionString* is available at run time in the appropriate config file.

Dynamic connection strings

DevForce also allows you to omit connection information from the run time configuration altogether and supply the connection string dynamically, using the *IDataSourceKeyResolver* interface.

DevForce actually uses its own implementation of the *IDataSourceKeyResolver* called the *DefaultDataSourceKeyResolver* to perform the key lookup we described above. You can implement a key resolver to override the default processing.

The *IDataSourceKeyResolver* interface defines a single method, *GetKey*, which you'll implement to provide your own key resolution logic. You should return either an in-memory *EdmKey* or *ClientEdmKey*, or null if you want default key resolution to be performed.

```
IDataSourceKey GetKey(String keyName, String keyExtension, bool onServer);
```

```
IDataSourceKey GetKey(String keyName, String keyExtension, bool onServer)
```

Note that the *EdmKey* you return is constructed only in code and does not need to be defined in the configuration file.

Although the key resolver is called on both client and server, key resolution is required on only the server. If you implement a custom resolver it's not required that you deploy your custom class to the client.

Anatomy of a connection string

Regardless of how you specify your connection information - whether in the *connectionStrings*, *edmKeys* or dynamically - you must still provide a valid Entity Framework connection string.

Here's an entry defined in the *connectionStrings* section:

```
<connectionStrings>
  <add
    name="NorthwindIBEntities"
    connectionString="
        metadata=res://*/DomainModel.csdl|res://*/DomainModel.ssdl|res://*/DomainModel.msl;
        provider=System.Data.SqlClient;
        provider connection string=
          &quot;
            Data Source=.;
            Initial Catalog=NorthwindIB;
            Integrated Security=True;
            MultipleActiveResultSets=True;
          &quot;"
    providerName="System.Data.EntityClient" />
</connectionStrings>
```

The <connectionString> has three parts:

- **name** - We discussed the name and how DevForce performs lookup above.
- **connectionString** - The Entity Framework connection string, also consisting of three parts:
    - **metadata** - The location of the metadata artifact files. Note above that an EDMX named "DomainModel" was used.
    - **provider** - The database provider to use for persistence operations (here it is SQL Server).
    - **provider connection string** - The provider-specific connection information, usually indicating the server, database and login information.
- **providerName** - This is the EF provider, and is optional.

Here's the same connection information specified in an *EdmKey*:

```
<edmKeys>
  <edmKey
    name="NorthwindIBEntities"
    connection="
        metadata=res://*/DomainModel.csdl|res://*/DomainModel.ssdl|res://*/DomainModel.msl;
        provider=System.Data.SqlClient;
        provider connection string=
          &quot;
            Data Source=.;
            Initial Catalog=NorthwindIB;
            Integrated Security=True;
            MultipleActiveResultSets=True;
          &quot;"
      />
</edmKeys>
```

As you can see, the *connection* contains the same information as the *connectionString* above.

Note that the sample SQL Server connection string above is more common to a developer's machine than a production deployment.
For more information on connection strings in EF, see http://msdn.microsoft.com/en-us/library/cc716756.aspx.

Security and connection strings

If you specify your connection information in a .config file, you may want to protect those sections containing sensitive information. You can encrypt configuration sections, such as the *connectionStrings* and *ideablade.configuration* sections, to limit unauthorized viewing. See here for more information. (To encrypt the ideablade.configuration section, make sure that the section definition contains the fully-qualified assembly name.)

In a production deployment, be sure to use a database account with appropriate privileges. Don't use an administrative account, and if you do use integrated security make sure that the service account does not have administrative privileges.

Also consider encrypting database traffic. See here for more information on encrypted SQL Server connections.