Contents

- EntityAspect should be ignored
- Check for an EntityServerError
- Metadata "Build Action" must be "Embedded Resource"
- Runtime exception that mentions metadata
- Failed to generate metatdata
- Metadata file regenerates for no apparent reason
- Exception: No datasource key found ...
- Exception on first operation AND "_" in DataSourceKeyName
- Exception after adding DbContext or changing its name
- Can't get the mapping right
- Works with the design database but not in production
- Missing the PostSharp library
- PostSharp library required in WPF application project
- PostSharp library required to Blend a Silverlight application project
- Error loading PostSharp symbol file
- PostSharp is not introduced in the build process
- Silverlight model project won't compile
- Invalid "Data Provider" in the connection string
- Missing property values in the database
- The data contract type 'MyNamespace.MyClass' cannot be deserialized...
- Missing SQL Server Express
- Careful with the DbContext constructor
- PostSharp compiler warnings
- MethodAccessException accessing OnSerializing method
- Using SQL 2005
- Build errors in Visual Studio 2012
- Runtime exception "A connectionString was not found for Key"
- Warning "CodeFirst entities were found in this assembly, but an EntityManager or DbContext was not"
- A few notes on PostSharp 3
- When all else fails ...

This topic identifies **common pitfalls** in DevForce Code First development and what to do about them.

Few human endeavors proceed perfectly. Code First development has its share of traps and tribulations. This topic covers some of the more frequently encountered problems and annoyances.

EntityAspect should be ignored

A build-time error message that reads, in part,

> EntityType 'EntityAspect' has no key defined. Define the key for this EntityType.

means you defined a custom *DbContext* but you neglected to ignore the DevForce *EntityAspect* type in your *DbContext*.

Check for an *EntityServerError*

A runtime exception thrown on the server-side *EntityServer* is propagated back to the client's *EntityManager*. DevForce then re-throws it as an *EntityServerException*.

Sometimes the client-side environment swallows this exception; Windows Presentation Foundation (WPF) is notorious in this regard. It's always a good idea to attach an event handler to the *EntityManager.EntityServerError* and give yourself a place to set a breakpoint.

Adding such a handler is good practice in any case

Here's an example from the "CodeFirstWalk" sample code:

```
private void SetupManager()
 {
    Manager = new ProductEntities();
    Manager.EntityServerError += (s, e) => Log("Server error:" + e.Exception.Message);
 }
}
```

You can set the breakpoint inside the lambda expression or inside the *Log* method.

Metadata "Build Action" must be "Embedded Resource"

When refactoring your model (e.g., breaking out a model project from your web project), you will relocate the ***ibmmx* metadata file** to the new model project. Make sure that the *ibmmx* file **"Build Action" in the Property Window is set to "Embedded Resource"**.

If you neglect this step, you may encounter a parade of bizarre complaints (null reference exception, "can't call an internal ctor", etc.) when running in n-tier mode (e.g., in Silverlight). The root cause: although you see the *ibmmx* file in your project, DevForce can't find it at runtime.

Runtime exception that mentions metadata

A DevForce entity metadata file (a file ending in ".ibmmx") is supposed to re-generate when you change your entity model classes and rebuild the project.

Maybe it didn't. See if DevForce failed to generate metadata

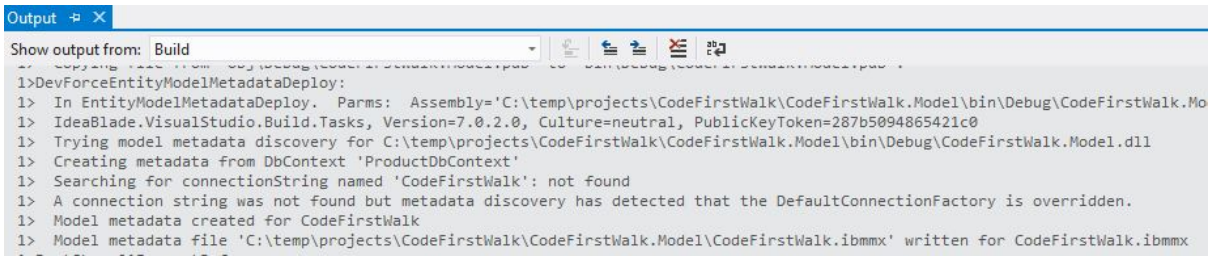
Failed to generate metatdata

If is always safe to delete a metadata file and re-build the project. The build process should regenerate it. If you **do not see** a new metadata file in your project, you know that DevForce was unable to create the metadata for some reason.

Open the Visual Studio Output Window and show the output from the "Build" process.

Make sure you've configured Visual Studio to provide enough detail. The "MSBuild project build output verbosity" should at least be "Normal". You can navigate to this setting in Visual Studio as follows: Tools | Options | Projects and Solutions | Build and Run.

Your log messages should look something like these:



In this example, a connection string was not found, so the task writes an informational message alerting you to the convention currently in place. See the topic on setting the database connection if you think it is important for metadata discovery to find your existing database.
The most important lines are at end of the output window where it tells you if metadata were created successfully:

> Model metadata created for CodeFirstWalk
> Model metadata file ....\CodeFirstWalk.ibmmx' written for CodeFirstWalk.ibmmx

or

> Model metadata created for CodeFirstWalk
> Model metadata for CodeFirstWalk.ibmmx is unchanged

If you see something to the contrary, metadata generation failed. The explanation for the failure - and what to try next - should appear in the Output window as well.

Metadata file regenerates for no apparent reason

Although neither you nor any other member of the team has made a change to the model, the metadata are regenerated when you build the application. You notice this because your source control system tells you that the *ibmmx* file has pending changes.

The probably cause is that your DevForce license key is not the same one that was used when the metadata were last generated and checked in. In the current DevForce release, a different license key is one reason to rebuild the metadata.

There are a couple of workarounds.

1. You can revert the change with your Source Control Tool; the annoyance remains of course.

2. Get everyone on the team to use the same license key. You can reset your key with "*Windows | All Programs | DevForce 2010 v.... | Tools | Product Key Updater*" which launches a tool (admin rights required) that accepts a new key.

For a future release, we are considering an option that would prevent regeneration of metadata when the only difference is a license key change.

That begs the question of how to force regeneration if you actually ***want*** to regenerate. That's easy: delete the *ibmmx* file and rebuild; it will be regenerated as part of the build process.

Exception: No datasource key found ...

You pressed [F5-Start Debugging] shortly after making a change to the model or perhaps after deleting the metadata file to force it's regeneration. The application compiled without error and started. But it crashed as soon as it made the first query. The exception, if you were lucky enough to catch it, was "*No datasource key found for this entity type: ...*".

Yet if you stop debugging and press [F5] again, it compiles, runs, ... and works. What's going on?

What's going on is: you must **build twice whenever the metadata file changes**!

As explained in the metadata generation topic, DevForce generates the metadata file during the first build, after the initial compilation. It is too late to include the file in the model assembly as an embedded resource. The metadata file will be included in the assembly during the second build.

The evidence is in the Output window when you show the Build output. The first [F5] yields compilation #1 whose last output message reads: "*Model metadata file '...' written for SomeFilename.ibmmx*". The second [F5] produces compilation #2 whose last output message reads: "*Model metadata for SomeFilename.ibmmx is unchanged*".

The "*...ibmmx is unchanged*" message is what you're looking for. So remember to build the project first .... and then press [F5].

Exception on first operation AND "_" in *DataSourceKeyName*

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

Exception after adding DbContext or changing its name

You might have two metadata ".ibmmx" files with conflicting model definitions.

The most likely cause: you added a new *DbContext* or renamed an existing *DbContext.*

The name of the metadata ".ibmmx" file corresponds to the DataSourceKeyName of your model. Changing the name results in generation of a newly-named metadata file, and removal of the old file. If the old file was not removed, when DevForce discovers two conflicting metadata files at runtime it throws an exception.

Solution: check for an obsolete metadata ".ibmmx" file and delete it. You can always delete all metadata ".ibmmx" files and re-build the project to generate the one true metadata file.

Can't get the mapping right

You may struggle to produce the correct mapping between entities and database objects. The error messages at build or run time are usually helpful in finding the problem but don't always provide sufficient guidance on how to do what you intended to do.

It's always good to have an idea about what has changed recently. Did you or someone else modify the model? Take note of the classes and associations that changed. Did you or someone else modify the design database? Take note of the tables, columns, and foreign key relationships that changed.

Then you might ask DevForce code generation for a little help. In a clean solution, try building a model using DevForce "Code Second". You can build the entire model or just focus on the suspect classes and associations. Compare the entity classes that DevForce generates with the entity class you wrote. You may find helpful clues in the way DevForce defined its mappings.

Works with the design database but not in production

Are you sure that the design database and the production database have exactly the same schema? Customers frequently tell us that they are "sure" ... only to discover that someone else has made a change they didn't know about. Have you confirmed that the two schemas are the same using automated tools? 3rd party vendor, redgate offers highly regarded products that are well worth their price.

Check also that your application model assemblies on the server are the same assemblies you are using in development. Again, customers frequently tell us that they are "sure" ... only to discover that someone else has deployed different assemblies or that the proper assemblies were not deployed. Make sure you update your application version numbers every time you build. A good Continuous Integration practice can help.

Missing the PostSharp library

You'll get the following build error if you neglected to reference the PostSharp.dll (or PostSharp.SL.dll in Silverlight) in your model project(s):

> Cannot resolve dependency to assembly 'PostSharp, Version=2.1.0.0, Culture=neutral, PublicKeyToken=b13fd38b8f9c99d7' because it has not been preloaded

Also make sure you deploy the PostSharp.dll with your application.

PostSharp library required in WPF application project

In a WPF application project which references your model project you may see a compiler warning from Microsoft.WinFX.targets stating: "Unknown build error, 'Cannot resolve dependency to assembly 'PostSharp'...'".

True, your Code First model is not in the application project. Unfortunately, WPF won't follow dependent assemblies during XAML compilation and therefore complains. Add the PostSharp assembly reference and this problem goes away.

But then you get a PostSharp warning about performance because PostSharp realizes it has nothing to process. You can disable PostSharp for this project.

PostSharp library required to Blend a Silverlight application project

The PostSharp library isn't required to **run** a Silverlight application. But it is required if you use Expression Blend to design Silverlight views. Blend uses WPF XAML compilation under the hood and has the same problem probing referenced assemblies that we described above. The resolution is the same as well: add a reference to *PostSharp.SL.dll* and then disable PostSharp processing of this project if (as expected) you aren't using AOP with any of this project's classes.

Note also that Blend works **only with the "Debug" configuration**. If you build the application in Visual Studio with a different configuration, you may have to re-build it in the "Debug" configuration. You can do that in Blend itself from the "Project" menu.

Error loading PostSharp symbol file

While re-building you get the following Build error:

> Error loading PostSharp symbol file: Access to the path 'C:\Users\...\Documents\Visual Studio 2012\Projects\... \Model.SL\Bin\Debug\Model.SL.pssym' is denied.

Among the likely causes: you deleted solution binaries in order to clear out lingering out-of-date assemblies. Unfortunately, the PostSharp process becomes confused and prevents you from building the model assembly.

Workaround: Close and re-open the solution.

PostSharp is not introduced in the build process

You'll get this error after using NuGet Package Restore to restore the *DevForce 2012 Code First* package, which also installs PostSharp.

The fix is to do as the error message states, and build the solution one more time.

Silverlight model project won't compile

Is the compiler complaining about a **class derived from *DbContext***? *DbContext* is an Entity Framework component that doesn't exist in Silverlight. It shouldn't be in your Silverlight model project either; delete the file or file link from your **Silverlight** project.

Invalid "Data Provider" in the connection string

A build-time error that reads:

Unable to find the requested .Net Framework Data Provider. It may not be installed.

indicates that the *providerName* on the *connectionString* is either invalid or is the EntityFramework EDM provider "System.Data.EntityClient" used with "Database First" or "Model First". You were probably migrating from a Database First model to a Code First model and overlooked this step while converting the EDM connection string to a normal database connection string.

You can remove the *providerName* attribute to allow it to default to the default value, “System.Data.SqlClient” or you can set it to a valid database provider name. Remember that this *connectionString* is in the *app.config* of the project you are building.

Missing property values in the database

You know that you are assigning a property but the value isn't showing up in the database. Check the access modifiers for your property's *get* and *set* methods. Are they *public*? If not, you must take extra configuration steps so that Entity Framework knows that you intend to map that control and to ensure that DevForce can serialize non-public data.

The data contract type 'MyNamespace.MyClass' cannot be deserialized...

You'll get this runtime exception if the Silverlight project holding your model does not contain an *InternalsVisibleTo* attribute to make entity "internals" visible to the *System.Runtime.Serialization* assembly.

Missing SQL Server Express

If you don't specify a connection string that Entity Framework can find, it may try to create a database that matches your entity model. It will attempt to create that database on SQL Server Express. You'll receive an error if SQL Express is not installed. You can change this default.

Careful with the DbContext constructor

At build time, DevForce will construct a new DbContext in order to obtain model metadata; at runtime, DevForce will construct a new DbContext for every query or save. This means the constructor on your custom DbContext class will be called at build time, and at run time for every query and save. For these reasons, you don't want to put logic here which should not be executed unintentionally or more than once. For example, instead of using a Database.CreateIfNotExists() call, which will connect to the database server to check for the existence of the database every time it's called, instead use a database initialization strategy, for example Database.SetInitializer(new DropCreateDatabaseIfModelChanges<MyDbContext>()). Better yet, place the initialization strategy in the static initializer for your custom DbContext.

PostSharp compiler warnings

In assemblies referencing your Code First model you may see a compiler warning from PostSharp like the following: "The module 'MyLib.dll' does not contain any aspect or other transformation.

For improved build-time performance, consider disabling PostSharp for this module by setting the compilation symbol (aka constant) 'SkipPostSharp' in your project, or set the MSBuild property 'SkipPostSharp=True'." The easiest way to resolve this is to go to the Properties page for your project and on the PostSharp tab set "Disable PostSharp" to "yes".

A WPF application project which references your model project may produce a compiler warning from Microsoft.WinFX.targets stating: "Unknown build error, 'Cannot resolve dependency to assembly 'PostSharp'...'". See discussion and resolution above.

MethodAccessException accessing OnSerializing method

You'll get this error in a Silverlight application when doing an *ImportEntities* call, using faking, or working with an *EntityCacheState* if you haven't granted *IdeaBlade.Core.SL* "friend" access to internals within your assembly. To fix the problem, add the following in the *AssemblyInfo* file in your model projects:

```
[assembly: InternalsVisibleTo("IdeaBlade.Core.SL,
PublicKey=0024000004800000940000000602000000240000525341310004000001000100b3f302890eb5281a7ab39b936ad9e0eded7c4a41abb440bead71ff5a31c
```

Using SQL 2005

DevForce automatically does a Refresh call after saving database changes. Unfortunately when using Code First with a SQL 2005 database this causes escalation of the *TransactionScope* to use a distributed transaction.

You can turn off DevForce's use of the *TransactionScope* by setting 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.

Build errors in Visual Studio 2012

1. *Could not obtain current project. Close other VS instances and try again.* In VS 2012 the build process is shelled to one or more MSBuild.exe instances, which confuses the DevForce Code First build time support. As the message states, if you close other VS 2012 instances and re-try the error should be resolved. This problem was fixed in release 7.0.1.
2. Building a solution with multiple Code First model projects might randomly result in the following error:
   The "EntityModelMetadataDeploy" task failed unexpectedly.
   System.Runtime.InteropServices.COMException (0x8001010A): The message filter indicated that the application is busy. (Exception from HRESULT: 0x8001010A (RPC_E_SERVERCALL_RETRYLATER))
   To avoid it, build each project separately. This problem was fixed in release 7.0.2.

Runtime exception "A connectionString was not found for Key"

You can receive this error if your model was compiled with a different version of DevForce. This can happen unintentionally if you have a version of DevForce 2010 earlier than 6.1.9 installed side-by-side with DevForce 2012. Side-by-side installation is not supported for versions of DevForce 2010 earlier than 6.1.9.

Warning "CodeFirst entities were found in this assembly, but an EntityManager or DbContext was not"

You might get this warning at build time if you have a *.cf marker file in a project without entity classes. But you also might get the warning when your EntityManager or DbContext can't be found. When this happens, it's usually because the version of the DevForce Code First package is different than the version of DevForce you are using. Check the Build Output window for DevForce messages as shown here and note the version of the IdeaBlade.VisualStudio.Build.Tasks assembly: it should be the same version as your other DevForce references.

A few notes on PostSharp 3

Application using a version of DevForce prior to 7.2 will use PostSharp version 2. If you manually install or upgrade the PostSharp NuGet package, be sure to specify the version number so that you don't mistakenly upgrade to version 3. More information is available here.

Applications using DevForce 7.2 or later will use PostSharp 3. The DevForce Code First package will have a dependency to a specific PostSharp 3 release: you should use the same version. For .NET applications, if you install a different version of the PostSharp NuGet package assembly binding redirects will be automatically added; this is not true for Silverlight and mobile projects.

When all else fails ...

You can post a message on the DevForce forum or contact Technical Support.

Before you do, you might try the following:

- Always check the *DevForceDebugLog.xml* to see if it is informative. The DevForce *EntityServer* typically writes it to a well-known location such as a *log* directory on the application's web site or in the web-or-application project's *bin/Debug* or *bin/Release* directories.
- Delete the metadata file(s) from the projects (the files with the .ibmmx extension); this is always safe.
- Delete the development test database from SQL Server **if and only if** they only contain seed and test data that you (or Entity Framework) can reconstruct.
- Make sure you've added an *EntityManager.EntityServerError* handler as described previously; set a breakpoint in the handler and see if it shows you an error. If it does but you can't decipher it, please include it in your forum post or support call.
- Clean the solution. You might also consider closing the solution and deleting the ***bin*** directories.