

## Contents

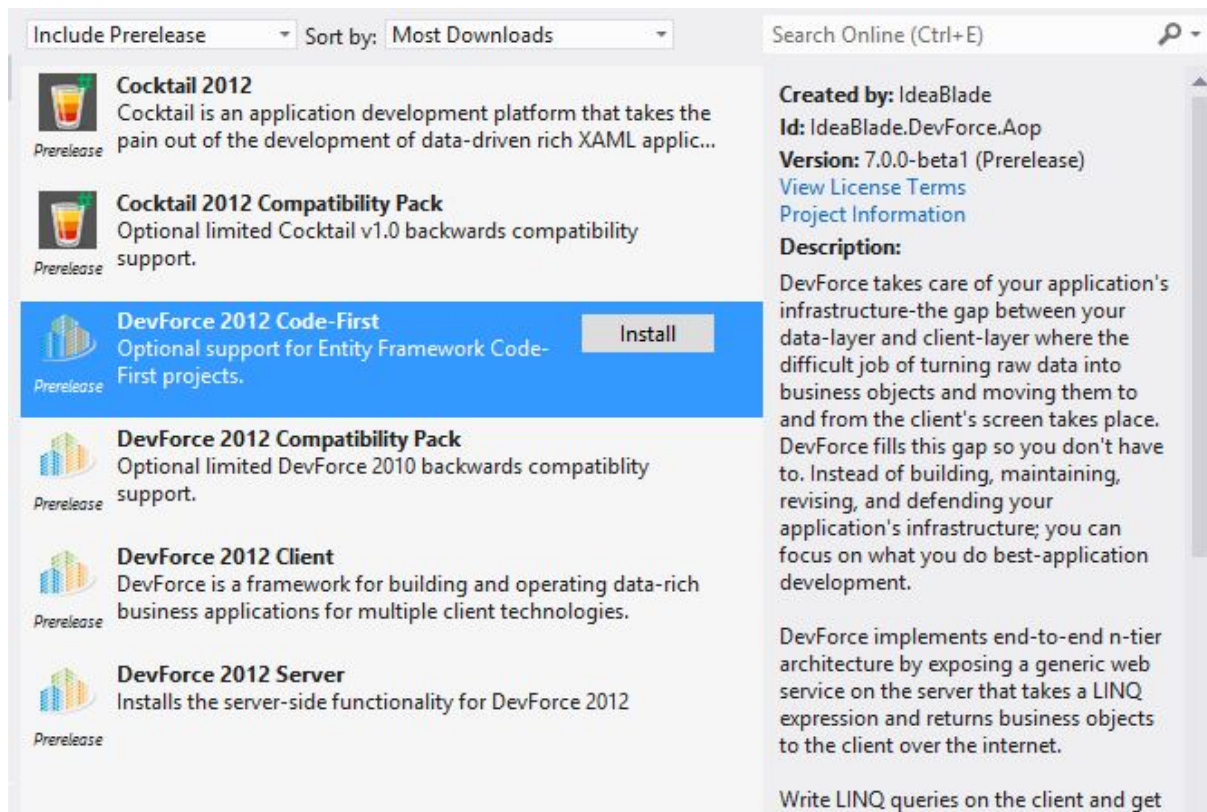
- [Two Steps to Metadata Generation](#)
- [Why Metadata](#)
- [Enable metadata generation with a .CF file](#)
- [Metadata file generation](#)
- [Metadata file and source control](#)
- [Metadata in Silverlight, Windows Store and Windows Phone projects](#)
- [The metadata generation process](#)
- [App.config & SQL Server Express](#)
- [Metadata on a build server](#)

This topic explains why DevForce needs **metadata**, how you enable DevForce metadata generation, and how DevForce generates metadata about your model from your entity classes, storing those metadata in an XML **project file with an ".ibmmx" file extension**.

A DevForce Code First model requires a companion entity metadata file that describes how the model works. This is an **XML file with an ".ibmmx" file extension**. DevForce (re)generates the metadata file automatically when you change your entity model classes. But only if you enable it.

## Two Steps to Metadata Generation

(1) Enable metadata generation by adding the *DevForce 2012 Code First* NuGet package to your project.



The package adds the required assembly references *IdeaBlade.Aop* and *PostSharp*, as well as the "Code First marker file" *DevForce.cf*, and modifies the MSBuild processing to add the *EntityModelMetadataDeploy* task.

### (2) Build the project

The ".ibmmx" metadata file will be (re)generated when the project is built successfully, and added to the assembly as an embedded resource.

You now know the only two things you have to *do* - perhaps the only things you *need to know* - about Code First metadata. The balance of this topic explains metadata and metadata generation in more detail.

## Why Metadata

DevForce needs metadata about your entity model **on the client** as well as the server. Metadata tells DevForce which properties are primary keys and whether they are generated by the database or by your client code. Metadata tells DevForce that the *Orders* property in the expression, *myCustomer.Orders*, is a navigation property that should return a collection of a particular customer's *Order* entities. Metadata tells DevForce that each of those *Order* entities has a *CustomerId* foreign key property with the value of the parent customer key. So informed, DevForce can seek those orders in the entity cache and return them if found or otherwise query for them. If it must issue a query, metadata tells DevForce how to compose that query on the client before sending it to the *EntityServer*.

Notice that all of the **metadata is about the entity model**. None of it concerns how the data are stored in the database. None of it relates to or depends upon the Entity Framework. DevForce does not require Entity Framework components on the client. That is one reason **DevForce entity classes can be compiled and consumed in Silverlight and mobile environments**.

## Enable metadata generation with a .CF file

DevForce (re)generates metadata only if the project includes a Code First marker file. The marker file can be any kind of file as long as it has a ".cf" file extension. By convention the marker file is a text file named "DevForce.cf".

You can create it yourself or you can use the *DevForce 2012 Code First* NuGet package to add it for you.

The marker file is not used by DevForce at runtime and need not be deployed.

Metadata can be generated only from the .NET project containing the model. The generated metadata file, \*.ibmmx, is then linked into client projects as an embedded resource.

## Metadata file generation

As a result of installing the DevForce Code First package, a DevForce MSBuild task, *EntityModelMetadataDeploy*, is added to your .NET model project's build pipeline.

When you build a project with a Code First model and the build task detects the marker file, DevForce gathers metadata about your model and outputs an "IdeaBlade Model Metadata" XML file (AKA ".ibmmx" file). You can think of the ".ibmmx" file as a substitute for the conceptual entity metadata file in an [EDMX](#).

"ibmmx" is an acronym for "IdeaBlade Model Metadata Xml"

The DevForce build process gives your metadata files their proper names and includes them as embedded resources of your projects.

Because you may move files around on your own, it may be helpful to know the rules:

- The metadata file extension must be ".**ibmmx**". DevForce will search for metadata files by this extension only.
- The name of the generated metadata file will be the same as the *DataSourceKeyName* for the model. If your project had a *DbContext* named *ProductDbContext* without a *DataSourceKeyNameAttribute*, the metadata file would be named *ProductDbContext.ibmmx*. If you're using a *DataSourceKeyNameAttribute* named "CodeFirstWalk", then the metadata file would be named *CodeFirstWalk.ibmmx*.
- The metadata file must be included in the project as an **embedded resource**.

Once created, an .ibmmx file will not be re-created until your model changes.

It is **always safe to delete a metadata file and re-generate it** in the next build. In fact, that can be a useful sanity check when things seem amiss. The DevForce build process detects that the file is missing and re-generates it. If you don't see the new ".ibmmx" file, you know that metadata generation failed; you should look at the Visual Studio output window for messages that explain why.

## Metadata file and source control

The *ibmmx* metadata file is a generated file and is not editable. It is intimately tied to the model itself; if you change the model and the *ibmmx* independently you will cause errors that could be very hard to explain or debug.

Yes, multiple people can evolve the model independently. But they have to share the *ibmmx* file. We recommend the following procedure to all model developers:

- update the model from source control
- make your model changes
- build to regenerate the *ibmmx*
- update from source control **again** for safety

- if update changed anything, re-build to regenerate the *ibmmx*
- checkin both your model changes and the revised *ibmmx*

When using source control (as surely you do), please take care to prevent merging of the repository *ibmmx* with your working copy of the *ibmmx*. Usually you can configure your source control system to treat an *ibmmx* file as a binary (non-merged) file type.

A conflicted *ibmmx* file will fail at runtime and you cannot easily resolve the conflict by hand; don't try. If you encounter a conflict, simply delete the *ibmmx* and re-generate it by re-building the project.

## Metadata in Silverlight, Windows Store and Windows Phone projects

You need entity metadata in your client project but you don't generate metadata there.

Instead, you link to each metadata ".*ibmmx*" file in the full .NET project - just as you link to the entity class and *EntityManager(s)* source code files. Then you build the model-cum-metadata in your client project which compiles the original source code with reference to environment-specific assemblies.

DevForce will attempt to automatically link generated *.ibmmx* files to a client project, but as with code files, you may have to manually set the file linkages if DevForce is unable to determine an appropriate linked project.

The linked *.ibmmx* files should have a Build Action of **embedded resource** in your client project.

**Be sure to rebuild your client project** whenever the metadata file changes. When the metadata file changes DevForce is able to detect the change and automatically rebuild the .NET project holding the model, but can't do this for the linked project. To ensure that the client assembly contains an embedded resource for the modified *.ibmmx* file, always build it after making any model changes.

## The metadata generation process

DevForce relies on Entity Framework Code First to produce most of the metadata. During the build, DevForce creates an instance of your custom *DbContext* (or the DevForce *DbContext* if you didn't write your own) and extracts conceptual model metadata from the *DbContext*'s EDM. For this reason, your model project must reference the Entity Framework libraries (Silverlight developers: [see above](#)).

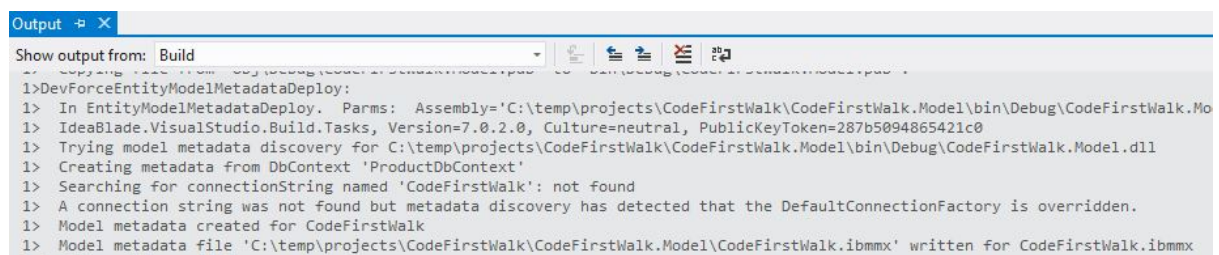
Because Entity Framework validates your model at **build time**, model validation errors - typically mapping inconsistencies - will surface as build errors. These errors are displayed in the Visual Studio Error List window.

The DevForce MSBuild task, *EntityModelMetadataDeploy*, writes its own messages to the Output window as it generates metadata and writes the ".*ibmmx*" file.

To see these message, 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.

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

Your log messages will look something like these:



```

Show output from: Build
1> DevForceEntityModelMetadataDeploy:
1> In EntityModelMetadataDeploy. Params: Assembly='C:\temp\projects\CodeFirstWalk\CodeFirstWalk.Model\bin\Debug\CodeFirstWalk.Model.dll'
1> IdeaBlade.VisualStudio.Build.Tasks, Version=7.0.2.0, Culture=neutral, PublicKeyToken=287b5094865421c0
1> Trying model metadata discovery for C:\temp\projects\CodeFirstWalk\CodeFirstWalk.Model\bin\Debug\CodeFirstWalk.Model.dll
1> Creating metadata from DbContext 'ProductDbContext'
1> Searching for connectionString named 'CodeFirstWalk': not found
1> A connection string was not found but metadata discovery has detected that the DefaultConnectionFactory is overridden.
1> Model metadata created for CodeFirstWalk
1> Model metadata file 'C:\temp\projects\CodeFirstWalk\CodeFirstWalk.Model\CodeFirstWalk.ibmmx' written for CodeFirstWalk.ibmmx
  
```

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 the end of the output window where it tells you if metadata was created successfully and if the *ibmmx* file was created or rewritten:

```

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.

Note that if your project contains other compilation errors, the *EntityModelMetadataDeploy* task will not be run.

## App.config & SQL Server Express

At **build time** when DevForce asks the *DbContext* for metadata, the Entity Framework will attempt to connect to the database to obtain storage model metadata. DevForce doesn't need the storage information for its metadata file, so there are three options you can choose from:

1. Have SQL Server Express installed and running ... OR ...
2. Set the [Database.DefaultConnectionFactory](#) for your preferred database provider ... OR ..
3. Add a configuration file (*app.config* or *web.config*) to your model project with a [connection string](#) for the *DataSourceKeyName* of the model.

When DevForce constructs the *DbContext* at build time, it ensures that a database isn't created by setting an appropriate [database initialization strategy](#).

## Metadata on a build server

**We don't recommend regenerating the ibmmx file on your build server.** The DevForce *EntityModelMetadataDeploy* MSBuild task will not successfully complete when running outside of Visual Studio. It will generate a warning message that the Visual Studio project cannot be found. Although this warning is harmless and your existing ibmmx file is left untouched, it's usually more efficient to disable metadata generation on your build server. You should build with the ibmmx metadata file(s) checked into your source control system.

You can prevent metadata generation by setting the *SkipDevForce* MSBuild property. There are several ways to do this.

1. Add the following to the MSBuild command line:
2. Create an *IdeaBlade.Override.targets* file containing the following:

```
/p:SkipDevForce=true
```

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">  
  <PropertyGroup>  
    <SkipDevForce>true</SkipDevForce>  
  </PropertyGroup>  
</Project>
```

The *EntityModelMetadataDeploy* task will look for this file in the same location as the *IdeaBlade.DevForce.Common.targets* file.

### You must not skip PostSharp processing on your build server!

Without PostSharp, your entities won't be enriched with the aspects required by DevForce and your application will not work. For more information on PostSharp on the build server, see [here](#).