The new Code First CTP 5 is out and there is a lot of buzz out there about it. It provides a different and in some ways easier way of using Entity Framework as your data layer. When I first saw a demo last summer I thought this is a really nice framework for generating EF. My next thought was that I would never actually hand-write code this way. Since all structure is in code, it is very easy to fat-finger and just plain old mess up. There is no model, so you potentially have a lot of developers writing a lot of classes. Every POCO needs to be hand-written and of course they will change over time. Each relation must be defined through either data annotations or with the fluent API. This is not really so bad with a few tables but could become quite unwieldy when you have hundreds (or more) of tables.

This is where the nHydrate framework comes in. nHydrate is model driven. Every aspect of the code is defined from a model. Using the standard EF there is a data model. Now this is not to be confused with model driven development or domain driven design (DDD). A data model defines data structure and EF provides a generated API on top of this, which is great. However this model is not at the solution level, which would allow you to create multiple projects off of a single model definition. There are tools, most notability T4 templates, that provide some of this functionality in an ad hoc fashion. I would argue that the nHydrate way of generating is a little better in that conceptually the model is at the solution level. Also generators can be added to the framework and loaded from a repository folder and used only when needed. One of my biggest grips with T4 is that my code changes all the time based on my data model. I have no way to control the auto generation. Also the template files must be inserted directly into my project as if they were source code.

The code first usage is lighter weight than a standard EF assembly. Not only is the assembly smaller but the classes are very much cleaner. Missing are base classes, interfaces, and complex data annotations, which is nicer because it does not cloud up the code. However the downside is you must write them yourself. This is my biggest issue with the new functionality. A developer must write tens of thousands of lines of code manually. Of course when you look at what is written, it is extremely repetitive code and begs to be generated.

Since nHydrate is simply a framework for managing generators, we can use the new code first generator with existing models. I took several models that I am already using in commercial products and re-generated them with the new code first templates. I was up and running ready to build a code first application off a database with hundreds of tables in about a minute.

Code first is simply a set of POCO objects that are used by a context object to interact with a database. The default samples simply have the POCO's mimic the database structure and all is well. However in the real world things are bit more complex. Often you want to map table and field names to different entities in code. All of this functionality has been handled by nHydrate for years and now you can leverage it with code first functionality. Since any non-trivial application will have objects that do not directly map to a database something must do the mapping. Using the new code first paradigms, you must define either an annotation on the POCOs or in the model created event. nHydrate uses the latter so the POCOs stay very clean. All the configuration and mapping code is in the OnModelCreating event.

In summary nHydrate allows you to use the newest EF incarnation while still maintaining a model driven architecture. In fact, you may need to use code first in conjunction with a classic EF assembly. Keeping the two in sync would be a nightmare manually. Through in database upgrades and maybe an IoC layer and you will need a lot of developer and QA resources to keep these layers in sync. Since nHydrate has all of these generators, you need do no such thing. Remember model driven development means that the model is your golden source not a database. Make whatever changes to the model you wish and re-generate the suite of projects. You are guaranteed to be in sync all the time. Even your database is versioned and upgraded. All of this adds up to a great way to keep your whole solution in order. When new language features come out, there is no need to rewrite your code. All you need to do is re-generate!