Fluent NHibernate wiki

Conventions

From Fluent NHibernate

Contents

Conventions are small self-contained chunks of behavior that are applied to the mappings Fluent NHibernate generates. These conventions are of varying degrees of granularity, and can be as simple or complex as you require. You should use conventions to avoid repetition in your mappings and to enforce a domain-wide standard consistency.

> Creating implementations of the interface convention types is always the recommended approach, but if you think otherwise these are available.

The Simplest Conventions

There are some situations that are so obvious that they just cried out for a simple shortcut that can be applied globally to your project:

```
Table.Is(x => x.EntityType.Name + "Table")
PrimaryKey.Name.Is(x => "ID")
AutoImport.Never()
DefaultAccess.Field()
DefaultCascade.All()
DefaultLazy.Always()
DynamicInsert.AlwaysTrue()
DynamicUpdate.AlwaysTrue()
OptimisticLock.Is(x => x.Dirty())
```

```
Cache.Is(x => x.AsReadOnly())
ForeignKey.EndsWith("ID")
```

Usage

To use these conventions you just need to call Add on the fluent configuration Conventions property. This method accepts a generics parameter of T, where T is constrained to IConvention interfaces.

```
Fluently.Configure()
  .Database(/* database config */)
  .Mappings(m =>
  {
    m.FluentMappings
      .AddFromAssemblyOf<Entity>()
      .Conventions.Add(PrimaryKey.Name.Is(x => "ID"));
  })
```

The Add method accepts a params array, so you can specify multiple conventions together.

```
.Conventions.Add(
  PrimaryKey.Name.Is(x => "ID"),
  DefaultLazy.Always(),
  ForeignKey.EndsWith("ID")
)
```

Inline Conventions

There are some shortcuts available for **conventions** that can be used inline (much like the old convention style); they're to be used for very simple situations where some people might think defining new types is overkill.

Builders

There is a ConventionBuilder class that you can use to define simple conventions. The general rule-of-thumb is if your convention has any logic, or spans multiple lines, then don't use the builders. That being said, they can be useful for simple scenarios.

The ConventionBuilder defines several static properties, one for just about all the available conventions. Through each property you can access two methods, Always and When. These can be used to create simple conventions that are either always applied, or only applied in particular circumstances.

Always

These inline conventions are applied to every mapping instance of their type, they're handy for catch-all situations.

```
ConventionBuilder.Class.Always(x => x.Table(x.EntityType.Name.ToLower()))
ConventionBuilder.Id.Always(x => x.Column("ID"))
```

When

These conventions are only applied when the first parameter evaluates to true.

```
ConventionBuilder.Class.When(
  c => c.Expect(x.TableName, Is.Not.Set), // when this is true
  x => x.Table(x.EntityType.Name + "Table") // do this
)
```

Writing Your Own Conventions

The conventions are built using a set of interfaces and base classes that each define a single method, `Apply`, with varying parameters based on the kind of convention you're creating; this method is where you make the changes to the mappings.

By default conventions are applied before your mappings are created from the fluent mappings. What this means is you define the "shape" of your mappings first, then the conventions are applied to set all default values, and finally any explicit changes you made with the fluent mappings are applied.

Given this mapping:

```
public class PersonMap : ClassMap<Person>
{
  public PersonMap()
  {
    Id(x => x.Id);
    Map(x => x.Name)
      .Column("FullName")
      .Length(100);
    Map(x => x.Age);
  }
}
```

The convention workflow is:

1. Determine the shape of your mapping by looking at what high-level methods you called. With the `PersonMap` example, the shape is considered to be: ClassMap of Person, with an Id, and two properties (Name and Age, respectively). The `Column` and `Length` calls do not form a part of this shape and are purely value-altering calls; these are ignored for the time being.
2. Conventions are then applied to this shape, setting any values specified, such as column names, lengths, cascades, access strategies, *etc.*
3. Finally, the value-altering methods from the fluent mappings are applied. In this case, the `Column` and `Length` methods will be applied *to the re-shaped mapping as defined by the convention*.

Your first convention

One of the most common conventions you'll use is for classes, and we'll start with that as an example. Class conventions are used to alter any value properties on the `ClassMap<T>` itself. If you wanted to alter specific properties or relationships, you'd use their respective conventions from the available interfaces and base classes.

```
public class LowercaseTableNameConvention : IClassConvention
{
  public void Apply(IClassInstance instance)
  {
    // alterations
  }
}
```

That's an empty class convention! Lets make it do something useful. For a simple example we'll set a default table name format for our entities: we'll follow some daft archaic rule that all table names must be prefixed with `tbl_`.

```
public void Apply(IClassInstance instance)
{
  instance.Table("tbl_" + instance.EntityType.Name);
}
```

Simple. The `IClassInstance` interface contains several properties that allow you to inspect what has been set on the mapping, and usefully gives you access to the underlying `System.Type` that the mapping is using. We can then use that information to change the table name using the `Table` method.

Conditional application of conventions

See: Acceptance criteria

Sometimes it may be necessary to control when a convention is applied; you may want a convention to only act on a subset of the mappings that it would normally alter. You can do this by implementing an additional interface that exposes an `Accept` method which allows you to define the criteria for which it will be applied.

The additional interface you need to implement has the same name as your main convention interface, with "Acceptance" on the end. So `IClassConvention` would have an `IClassConventionAcceptance` interface.

```
public class LowercaseTableNameConvention : IClassConvention, IConventionAcceptance<IClassInspector>
{
  public void Accept(IAcceptanceCriteria<IClassInspector> criteria)
  {
    // alterations
  }

  /* snip */
```

The `IAcceptanceCriteria` instance is what you use to set up the expected conditions that must be met before the convention is applied. For more information about the criteria you can create, see: Acceptance criteria.

```
public void Accept(IAcceptanceCriteria<IClassInspector> criteria)
{
  criteria.Expect(x => x.Type != typeof(SomeSpecialCase));
}
```

That example simply changes the convention so that it won't be applied to any mappings of the `SomeSpecialCase` type.

Not specifying any expectations is the same as not implementing `IConventionAcceptance<>`.

Configuration

Now that we've created our convention, we need to inform Fluent NHibernate of how to use it. The simplest way to do this is just to use the Fluent configuration mapping setup to use all conventions in an assembly. Alternatively, you can add individual conventions piecemeal. Both these are done through the `Conventions` property of the fluent mappings setup.

```
Fluently.Configure()
  .Database(/* database config */)
  .Mappings(m =>
  {
    m.FluentMappings
      .AddFromAssemblyOf<Entity>()
      .Conventions.AddFromAssemblyOf<LowercaseTableNameConvention>();
  })
```

Alternatively, you can call `Add` to add an instance or a type, or you can use the `Setup` method to call a lambda that can be used to supply multiple different conventions (e.g. some from an assembly, and some individually).

Once you've set that up, Fluent NHibernate will automatically call your convention when it generates the mappings.

Note on previous conventions

If you were using our previous conventions API, then you may be interested in converting to new style conventions, which explains the various differences you may encounter; however, there's nothing difficult about it, you just need to pick the right interface. If you're not quite comfortable with this change yet, the convention shortcuts might be of some use.

Retrieved from "http://wiki.fluentnhibernate.org/Conventions"

Page Discussion View source History
