Entity Framework - Is there a way to automatically eager-load child entities without Include()?

Asked 6 years, 7 months ago Active 1 year ago Viewed 37k times

67

17

Is there a way to decorate your POCO classes to automatically eager-load child entities without having to use `Include()` every time you load them?

Say I have a Class Car, with Complex-typed Properties for Wheels, Doors, Engine, Bumper, Windows, Exhaust, etc. And in my app I need to load my car from my DbContext 20 different places with different queries, etc. I don't want to have to specify that I want to include all of the properties every time I want to load my car.

I want to say

```
List<Car> cars = db.Car
                .Where(x => c.Make == "Ford").ToList();

//NOT .Include(x => x.Wheels).Include(x => x.Doors).Include(x => x.Engine).Include(x =>
x.Bumper).Include(x => x.Windows)


foreach(Car car in cars)
{

//I don't want a null reference here.

String myString = car.**Bumper**.Title;
}
```

Can I somehow decorate my POCO class or in my `OnModelCreating()` or set a configuration in EF that will tell it to just load all the parts of my car when I load my car? I want to do this eagerly, so my understanding is that making my navigation properties virtual is out. I know NHibernate supports similar functionality.

Just wondering if I'm missing something. Thanks in advance!

Cheers,

I like the solution below, but **am wondering if I can nest the calls to the extension methods.** For example, say I have a similar situation with Engine where it has many parts I don't want to include everywhere. Can I do something like this? (I've not found a way for this to work yet). This way if later I find out that Engine needs FuelInjectors, I can add it only in the BuildEngine and not have to also add it in BuildCar. **Also if I can nest the calls, how can I nest a call to a collection? Like to call BuildWheel() for each of my wheels from within my BuildCar()?**

```
public static IQueryable<Car> BuildCar(this IQueryable<Car> query) {
     return query.Include(x => x.Wheels).BuildWheel()
                 .Include(x => x.Doors)
                 .Include(x => x.Engine).BuildEngine()
                 .Include(x => x.Bumper)
                 .Include(x => x.Windows);
}

public static IQueryable<Engine> BuildEngine(this IQueryable<Engine> query) {
     return query.Include(x => x.Pistons)
                 .Include(x => x.Cylendars);
}

//Or to handle a collection e.g.
 public static IQueryable<Wheel> BuildWheel(this IQueryable<Wheel> query) {
     return query.Include(x => x.Rim)
                 .Include(x => x.Tire);
}
```

Here is another very similar thread in case it is helpful to anyone else in this situation, but it still doesn't handle being able to make nexted calls to the extension methods.

Entity framework linq query Include() multiple children entities

entity-framework-4.1 entity code-first entity-framework-4.3

edited May 23 '17 at 12:34
Community ♦
1 1

asked Jan 24 '13 at 22:49
Nathan Geffers
338 1 3 5

Nathan, very interesting challenge. Please explain why you want/need to avoid using active loading techniques like include(). – Dave Alperovich Jan 24 '13 at 23:12

I want to avoid this because I have a complex model, but very few records, so I want the data to load eagerly, but don't want to miss any includes of grandchildren anywhere in the app. – Nathan Geffers Jan 25 '13 at 16:41

Related question/duplicate: stackoverflow.com/questions/5001311/… – Stefan May 2 '17 at 13:41

2 Answers

61

No you cannot do that in mapping. Typical workaround is simple extension method:

```
public static IQueryable<Car> BuildCar(this IQueryable<Car> query) {
     return query.Include(x => x.Wheels)
                 .Include(x => x.Doors)
                 .Include(x => x.Engine)
                 .Include(x => x.Bumper)
                 .Include(x => x.Windows);
}
```

Now every time you want to query `Car` with all relations you will just do:

```
var query = from car in db.Cars.BuildCar()
            where car.Make == "Ford"
            select car;
```

Edit:

You cannot nest calls that way. Include works on the core entity you are working with - that entity defines shape of the query so after you call `Include(x => Wheels)` you are still working with `IQueryable<Car>` and you cannot call extension method for `IQueryable<Engine>`. You must again start with `Car`:

```
public static IQueryable<Car> BuildCarWheels(this IQuerable<Car> query) {
    // This also answers how to eager load nested collections
    // Btw. only Select is supported - you cannot use Where, OrderBy or anything else
    return query.Include(x => x.Wheels.Select(y => y.Rim))
                .Include(x => x.Wheels.Select(y => y.Tire));
}
```

```
public static IQueryable<Car> BuildCar(this IQueryable<Car> query) {
     return query.BuildCarWheels()
                 .Include(x => x.Doors)
                 .Include(x => x.Engine)
                 .Include(x => x.Bumper)
                 .Include(x => x.Windows);
}
```

The usage does not call `Include(x => x.Wheels)` because it should be added automatically when you request eager loading of its nested entities.

Beware of complex queries produced by such complex eager loading structures. It may result in very poor performance and a lot of duplicate data transferred from the database.

edited May 23 '17 at 11:54 — Community ♦ — 1 1

answered Jan 25 '13 at 11:31 — Ladislav Mrnka — 333k 56 625 648

That's exactly what I need! Thanks! – Nathan Geffers Jan 25 '13 at 14:21

Could I nest calls to these extension methods for child entities? (See updated question) – Nathan Geffers Jan 25 '13 at 16:07

1 Having an [Include] attribute gets my vote! (see link in the answer) – Nathan Geffers Jan 25 '13 at 17:35

Check my edited answer. – Ladislav Mrnka Jan 28 '13 at 9:07

So then this means I still need to write a separate method about loading rims and tires whenever I need to load busses, cars, trucks etc. I'll just include them in my LoadBus, LoadTruck, LoadCar methods, but I was trying to avoid having those methods need to know anything about the children of Tire. Too bad, but at least I have my answer. Thanks! – Nathan Geffers Jan 28 '13 at 14:14

3

Had this same problem and saw the other link which mentioned an `Include` attribute. My solution assumes that you created an attribute called `IncludeAttribute`. With the following extension method and utility method:

```
public static IQueryable<T> LoadRelated<T>(this IQueryable<T> originalQuery)
{
    Func<IQueryable<T>, IQueryable<T>> includeFunc = f => f;
    foreach (var prop in typeof(T).GetProperties()
```

```
f.Include(prop.Name);
            includeFunc = Compose(includeFunc, chainedIncludeFunc);
        }
        return includeFunc(originalQuery);
    }

    private static Func<T, T> Compose<T>(Func<T, T> innerFunc, Func<T, T> outerFunc)
    {
        return arg => outerFunc(innerFunc(arg));
    }
```

edited Sep 18 '17 at 18:46

answered Sep 18 '17 at 17:31

Lunyx

1,691 4 20 42