Tapping into EF Core's Pipeline

Julie Lerman

EF Core Expert and Software Coach

@julielerman | thedatafarm.com

Module Overview

Overriding SaveChanges

Saving and tracking event handlers

Interceptors for database commands and more

Exploring ChangeTracker Entries for Overriding the SaveChanges Method

SaveChanges Method is Virtual

You can override SaveChanges in your DbContext class

Add logic just before or just after EF Core calls SaveChanges internally

Accessing the ChangeTracker EntityEntries

```
DbContext.ChangeTracker.Entries().ToList();
```

EF Core uses the info in these entries to construct its SQL command

```
public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  return base.SaveChanges(); //sends data to database
}

public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  int affected=base.SaveChanges(); //sends data to database
  //perform some other custom logic after DB calls
  return affected;
}
```

Override SaveChanges to Apply Custom Logic Around DB Call

Unless you want to completely override SaveChanges, call base.SaveChanges, which will send the commands to the database.

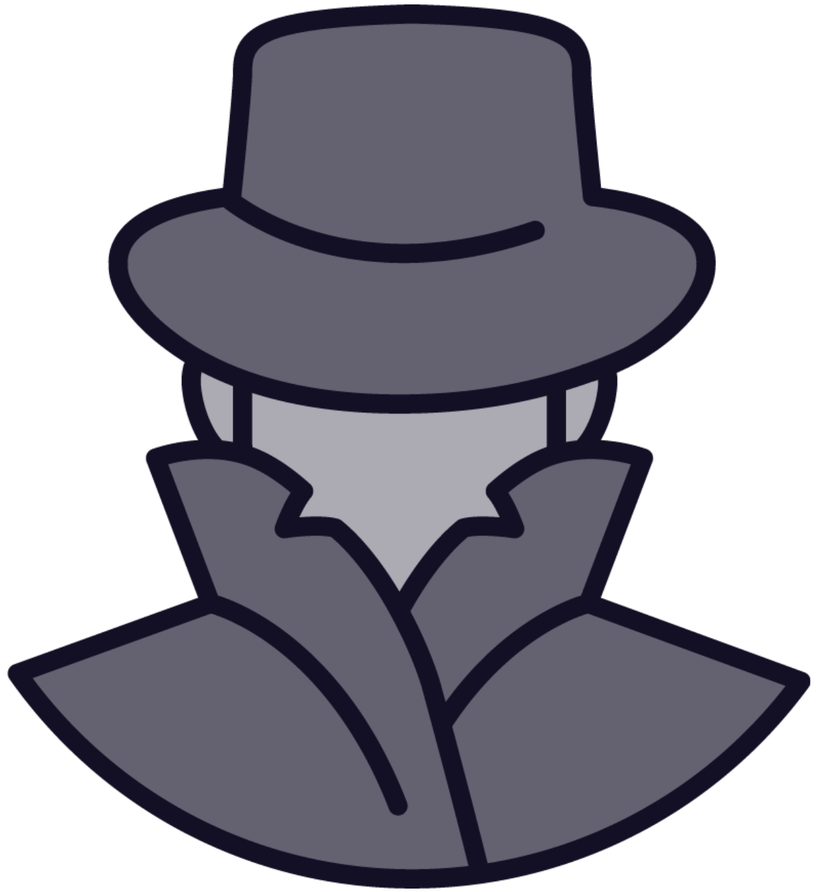
Please learn more than these fundamentals of ChangeTracker.Entries before altering them in your code

```
public override int SaveChanges()
{
  //perform your custom logic on ChangeTracker.Entries() or other data
  return base.SaveChanges(); //sends data to database
}
```

Override SaveChanges to Apply Custom Logic Around DB Call

Unless you want to completely override SaveChanges, call base.SaveChanges, which will send the commands to the database.

Updating Shadow Properties During SaveChanges



Why Shadow Properties?

Store data that's irrelevant to your biz logic

Examples:

- Row created/updated timestamps
- User who created/modified row

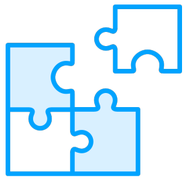
Useful for reports against the database such as auditing

Extraneous to business logic and would be in the way

Shadow Properties

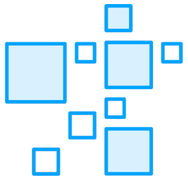
Defined by the DbContext and persisted into database

Use migrations to get the new property into the database



Not part of your class so use the DbContext to set values

Just before SaveChanges builds SQL is a great time to set values



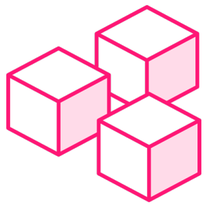
You can define a shadow property for all entities & update them, too

Use SaveChanges & Shadow Properties to Populate Audit Data

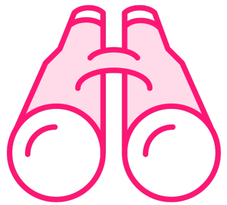
```
modelBuilder.Entity<Author>()
.Property<DateTime>("LastUpdated");


private void UpdateAuditData()
{
  foreach(var e in
    ChangeTracker.Entries<Author>())
  {
    entry.Property("LastUpdated")
         .CurrentValue =DateTime.Now;
  }
}

public override int SaveChanges()
{
  UpdateAuditData();
  return base.SaveChanges();
}
```

- Define shadow properties in OnModelCreating
- Create a method that updates shadow properties in the ChangeTracker.Entries()
- Call the new method in SaveChanges override
- Make sure internal SaveChanges is still called!

New to DbSet.Local: FindEntry() & GetEntries()

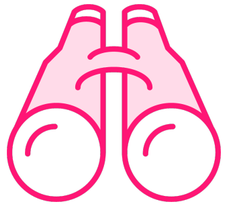


Returns entity objects, not EntityEntry objects



Lookup tracked entities by primary, alternate, or foreign key

```
context.Authors.Local.FindEntry(1)
```



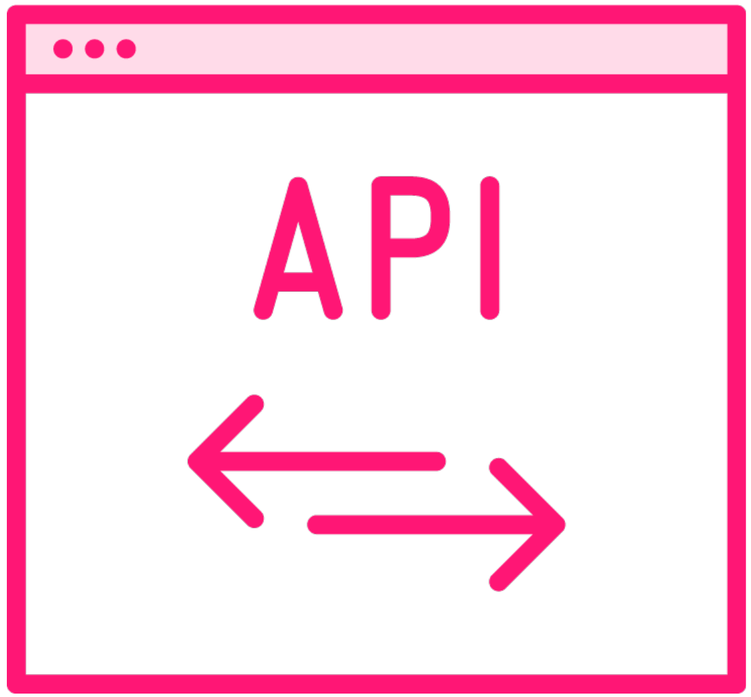
Lookup tracked entities by property value

```
context.Authors.Local.GetEntries(nameof(Author.FirstName), “Kim”)
```

More details and suggested use cases in EF Core 8 What’s New docs

Using EF Core Pipeline Events

Events Exposed in EF Core API




```
DbContext.SavingChanges

DbContext.SavedChanges

DbContext.SaveChangesFailed

ChangeTracker.Tracked

ChangeTracker.StateChanged
```

Implementing Event Handlers

A method to execute custom logic

Declare the method as a handler of the target event

```
1. private void UpdateAuditData()
  {
    foreach (var entry in ChangeTracker.Entries<Author)
    {      entry.Property("LastUpdated").CurrentValue =DateTime.Now;    }
  }

2. SavingChanges += SavingChangesHandler;

3. private void SavingChangesHandler(object sender, SavingChangesEventArgs e)
  {
    UpdateAuditData();
  }
```

Using SavingChanges & Shadow Properties to Populate Audit Data

SavingChanges occurs before the SaveChanges logic is performed

1. Create a method that updates shadow property in the ChangeTracker.Entries
2. Declare event hander in constructor
3. Add logic to event handler method

SaveChange Events vs. Override SaveChanges

Individual Event Handlers

```
private void SavingChangesHandler(...)
{
    //logic before save
}

private void SavedChangesHandler(...)
{
    //logic after save
}

private void SaveFailedHandler(...)
{
    //error handling
}
```

vs

Override SaveChanges

```
public override int SaveChanges()
{
    //some actions before save

    try
    {
       int affected=base.SaveChanges();

       //some actions after saved
       return affected;
     }
     catch
     {
       //save failed: error handling
     }
}
```

You could use these events to emit alternate information that isn't tracked by the logger.

Using Interceptors to Inject Logic into EF Core's Pipeline

If you used Interceptors in EF6, these work in a similar way.

Interceptors

Entity Framework Core (EF Core) interceptors enable interception, modification, and/or suppression of EF Core operations. This includes low-level database operations such as executing a command, as well as higher-level operations, such as calls to SaveChanges.

learn.microsoft.com/ef/core/logging-events-diagnostics/interceptors

Intercepting Database Operations

Interceptor	Database operations intercepted
IDbCommandInterceptor	Before sending a command to the database After the command has executed Command failures Disposing the command's DbDataReader
IDbConnectionInterceptor	Opening and closing connections Connection failures
IDbTransactionInterceptor	Creating transactions Using existing transactions Committing transactions Rolling back transactions Creating and using savepoints Transaction failures

Interceptors Go Beyond Database Activity

IgnoringIdentityResolutionInterceptor

IdentityResolutionInterceptor

InstantiationBindingInterceptor

MaterializationInterceptor

QueryExpressionInterceptor

SaveChangesInterceptor

SingletonInterceptor

SaveChangesInterceptor

UpdatingIdentityResolutionInterceptor

Setting Up Interception

```
internal class MyInterceptor :
 DbCommandInterceptor
{
  public override
   InterceptionResult<DbDataReader>
   ReaderExecuting(DbCommand command,
   CommandEventData eventData,
   InterceptionResult<DbDataReader> result)
   {
     //do something e.g., edit command
     return result;
   }
}

optionsBuilder.UseSqlServer(connString)
.AddInterceptors(new MyInterceptor());

builder.Services.AddDbContext
(options=> options
.AddInterceptors(new MyInterceptor());
```

- Define an interceptor class
- Register the interceptor in your DbContext
- Or via ASP.NET Core services configuration

Review

There are a variety of ways to tap into EF Core's pipeline

Unlike logging, these methods let you change data and behavior

ChangeTracker.Entries is a critical tool when affecting how SaveChanges works

Changing how EF Core's pipeline works is powerful but requires a good understanding to avoid problematic side effects

Thank you for watching this course!

Resources

EF Core Documentation: learn.microsoft.com/ef/core

Tapping into EF Core's Pipeline codemag.com/Article/2103051

What's New in EF Core 8 Docs (including FindEntry method)
learn.microsoft.com/ef/core/what-is-new/ef-core-8.0/whatsnew

Entity Framework Core Fundamentals

Julie Lerman

EF Core Expert and Software Coach

@julielerman | thedatafarm.com