1.– 4. September 2014
in Nürnberg

Herbstcampus

Wisse
par e

Home Improvement Entity Framework 6.x

Thomas Haug
MATHEMA Software GmbH

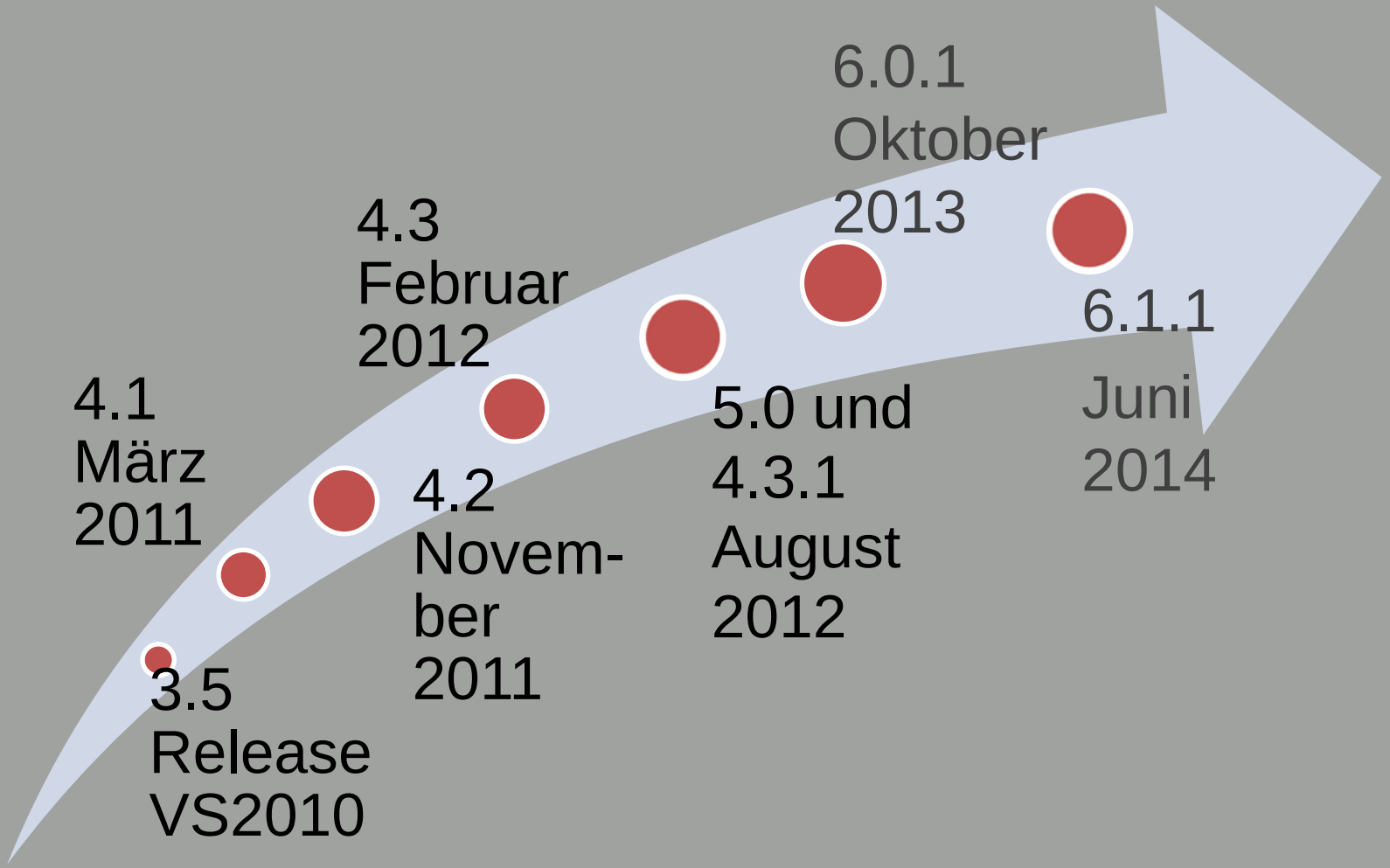
Agenda

- Enitity Framework 6.0 Features
- Enitity Framework 6.1 Features
- Entity Framework 6.x Erweiterungen
- Zusammenfassung

Agenda

- Enitity Framework 6.0 Features
- Enitity Framework 6.1 Features
- Entity Framework 6.x Erweiterungen
- Zusammenfassung

Entity Framwork - Versionen



Entity Framework – Features 6.0

- Async Query und Save Methoden (task-based async patterns)
- Custom Code First Conventions
- Code First Mapping für Insert, Update, Delete Stored Procedures
- Connection Resiliency für temporäre Verbindungsausfälle
- **Dependency Resolution zur Unterstützung des Service Locator Patterns**
- Code-Based Configuration
- **Interception/SQL logging**

Entity Framework – Features 6.0

- Konfigurierbare Migrations History Tabelle
- Mehrere Code First Modelle pro Datenbank über mehrere Context Objekte
- Neue Methode DbModelBuilder.HasDefaultSchema im Code First API
- Enums, Spatial und verbesserte Performance für .NET 4.0
- Bereits offene DbConnection kann an DbContext übergeben werden
- Vereinfachte Transaktionsaktionshandhabung mit DbContext.Database.UseTransaction und DbContext.Database.BeginTransaction

Entity Framework – Features 6.0

- Standard Transaktionsisolation für Code First auf READ_COMMITTED_SNAPSHOT umgestellt
- Performance Optimierungen für Enumerable.Contains in LINQ Queries
- Verbessertes Startverhalten, insbesondere für große Modelle
- DbModelBuilder.Configurations.AddFromAssembly für Fluent API mit Configuration Klassen
- Custom Migrations Operations
- Pluggable Pluralization & Singularization Service
- Verbesserte POCO Unterstützung

Entity Framework 6 - Namespaces

- System.Data.* wurde nach System.Data.Entity.Core.* „bewegt“

- Beispiele
  - System.Data.EntityException → System.Data.Entity.Core.EntityException
  - System.Data.Objects.ObjectContext → System.Data.Entity.Core.Objects.ObjectContext
  - System.Data.Objects.DataClasses.RelationshipManager → System.Data.Entity.Core.Objects.DataClasses.RelationshipManager

Entity Framework – Features 6.1

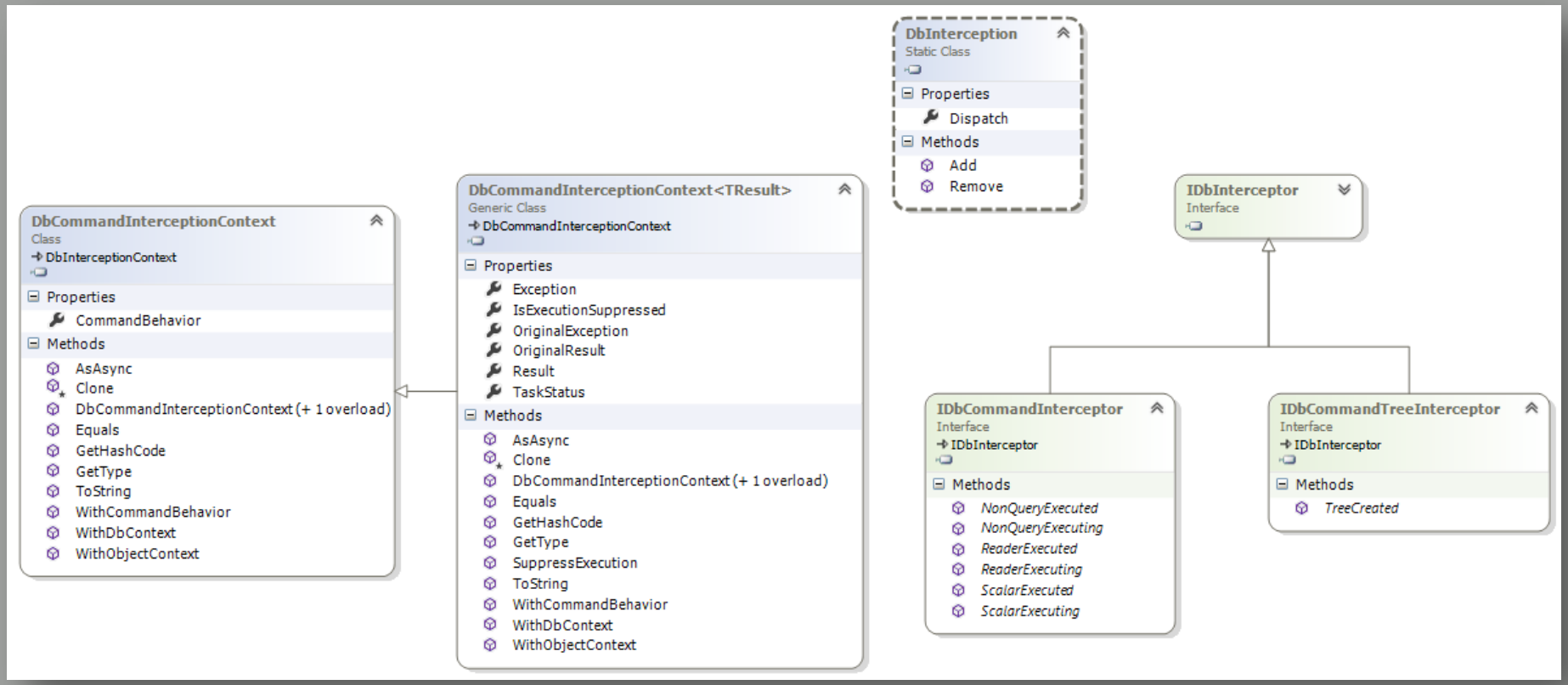
- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

Interception (EF 6.0)

- Einfaches „Framework“ um Zugriffe auf die Datenbank abzufangen
- Vor Aufruf des Datenbank-Layers
  - XyzExecuting(…)
- Nach Aufruf des Datenbank-Layers
  - XyzExecuted(…)
- Interceptoren werden über die statische Klasse DbInterception registriert bzw. deregistriert
- Werden in der Reihenfolge ihrer Registrierung ausgeführt
- Gelten App-Domain weit

Interception (EF 6.0)

- Einfaches „Framework“



Interception (EF 6.0)

- Beispiel

```
using System.Data.Common;

public class DefaultInterceptor : IDbCommandInterceptor {
   public virtual void NonQueryExecuted(DbCommand command,
               DbCommandInterceptionContext<int> interceptionContext) {}

   public virtual void NonQueryExecuting(DbCommand command,
                DbCommandInterceptionContext<int> interceptionContext){}

   public virtual void ReaderExecuted(DbCommand command,
       DbCommandInterceptionContext<DbDataReader> interceptionContext){}

   public virtual void ReaderExecuting(DbCommand command,
       DbCommandInterceptionContext<DbDataReader> interceptionContext){}

   public virtual void ScalarExecuted(DbCommand command,
             DbCommandInterceptionContext<object> interceptionContext){}

   public virtual void ScalarExecuting(DbCommand command,
             DbCommandInterceptionContext<object> interceptionContext){}
}
```

Interception (EF 6.0)

- Beispiel

```
public class MeinInterceptor : DefaultInterceptor {
   public override void ReaderExecuted(DbCommand command,
       DbCommandInterceptionContext<DbDataReader> intCtx) {
     Console.Write("ReaderExecuted Command: {0}",
                   command.CommandText);
   }
}
```

```
public class Program {
   static void Main(string[] args) {
      string con= @"…";
      DbInterception.Add(new MeinInterceptor());
      using (DemoContext ctx = new DemoContext(con)) {
        Category category = new Category() { Name = "TestCat"};

         ctx.Categories.Add(category);
         ctx.SaveChanges();
      }
   }
}
```

Dependency Resolution (EF 6.0)

- Ab EF 6.0 werden „wesentliche“ Dienste über einen Service Locator innerhalb des Frameworks identifiziert und genutzt
- Einsatz des Chain of Responsibilty Pattern
- IDbDepenendencyResolver
  - object GetService(Type type, object key);

- Verwendung

```
public class DemoConfiguration : DbConfiguration {
  public DemoConfiguration() {
     var conFactory = new MyConnectionFactory();
     var resolver = new
       SingletonDependencyResolver<IDbConnectionFactory>
       (conFactory)

     AddDependencyResolver(resolver);
    }
}
```

Dependency Resolution (EF 6.0)

- Folgende Services sind definiert
  - System.Data.Entity.IDatabaseInitializer<TContext>
  - System.Data.Entity.Migrations.Sql.MigrationSqlGenerator
  - System.Data.Entity.Core.Common.DbProviderServices
  - System.Data.Entity.Infrastructure.IDbConnectionFactory
  - System.Data.Entity.Infrastructure.IManifestTokenService
  - System.Data.Entity.Infrastructure.IDbProviderFactoryService
  - System.Data.Entity.Infrastructure.IDbModelCacheKeyFactory
  - System.Data.Entity.Spatial.DbSpatialServices
  - System.Data.Entity.Infrastructure.IExecutionStrategy
  - System.Data.Entity.Migrations.History.HistoryContextFactory
  - System.Data.Common.DbProviderFactory
  - System.Data.Entity.Infrastructure.IProviderInvariantName
  - System.Data.Entity.Core.Mapping.ViewGeneration.IViewAssemblyCache
  - System.Data.Entity.Infrastructure.Pluralization.IPluralizationService

Agenda

- Enitity Framework 6.0 Features
- Enitity Framework 6.1 Features
- Entity Framework 6.x Erweiterungen
- Zusammenfassung

Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

Interception (EF 6.1)

- Interceptoren können ohne Recompile der Anwendung aktiviert werden
  - Unterstützt Debugging in Produktivsystemen
- Entity Framework stellt eine Default-Implementierung eines Loggers bereit
  - schreibt in Console oder Datei
- Beispiel (mit Dateiausgabe)

```
<entityFramework>
    <interceptors>
        <interceptor type="System.Data.Entity.Infrastructure
            .Interception.DatabaseLogger, EntityFramework">
            <parameters>
                <parameter value="C:\Temp\LogOutput.txt"/>
            </parameters>
        </interceptor>
    </interceptors>
… </entityFramework>
```

Interception (EF 6.1)

- Beispielausgabe in Datei

```
INSERT [dbo].[__MigrationHistory]([MigrationId], [ContextKey], [Model],
 [ProductVersion])
VALUES (N'201409021006141_InitialCreate', N'EF6_Common.Context.BugShopContext',  …
-- Completed in 27 ms with result: 1

Committed transaction at 02.09.2014 12:06:15 +02:00
Disposed connection at 02.09.2014 12:06:15 +02:00

Opened connection at 02.09.2014 12:06:15 +02:00
Started transaction at 02.09.2014 12:06:15 +02:00
INSERT [dbo].[Category]([Cat_Name], [Cat_Descr])
VALUES (@0, NULL)
SELECT [CategoryID]
FROM [dbo].[Category]
WHERE @@ROWCOUNT > 0 AND [CategoryID] = scope_identity()
-- @0: 'TestCategory mit Interceptor' (Type = String, Size = 30)
-- Executing at 02.09.2014 12:06:15 +02:00
-- Completed in 5 ms with result: SqlDataReader

Committed transaction at 02.09.2014 12:06:15 +02:00
Closed connection at 02.09.2014 12:06:15 +02:00
```

Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

Handhabung Tx-Commit Fehler

- EF 6.0 führte das Feature „Connection Resiliency für temporäre Verbindungsausfälle“ ein

- Aber:

  Ursachen für Fehler bei Commit

  1. Commit -Ausführung im DB-Server schlägt fehl
  2. Commit auf DB-Server erfolgreich, aber aufgrund eines Netzwerkfehlers wird Commit Success nicht zur Anwendung übermittelt

  → Bei aktivem „Connection Resiliency“ kann im zweiten Fall ein wiederholter Commit passieren….

- Dieses kann zu Dateninkonsistenzen führen (siehe Issue #1114)

Handhabung Tx-Commit Fehler

- Das neue Feature versucht die potentiellen Fehler aus Szenario 2 zu verhindern
- Es wird eine Transaktionshistorie geschrieben

- Aktivieren über DbConfiguration

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;

public class MyConfiguration : DbConfiguration {
    public MyConfiguration()
    {
        SetExecutionStrategy(
            SqlProviderServices.ProviderInvariantName,
                () => new SqlAzureExecutionStrategy());

        SetTransactionHandler(
            SqlProviderServices.ProviderInvariantName,
                    () => new CommitFailureHandler());
}
}
```

Handhabung Tx-Commit Fehler

- Die Transaktionshistorie wird in die Tabelle „__TransactionHistory“ geschrieben

__TransactionHistory

	Column Name	Condensed Type	Nullable
🔑	Id	uniqueidentifier	No
	CreationTime	datetime	No

- Jede Transaktion wird in die Tabelle eingetragen und
- nach einem Commit entfernt

- Passiert ein Fehler beim Commit, wird geprüft, ob eine Zeile mit identischer Id bereits vorhanden ist
  - Falls ja, commit wurde noch nicht ausgeführt
    → SaveChanges() kann nochmals gerufen werden (retry)
  - Falls nein, commit wurde bereits ausgeführt (denn entsprechende Zeile fehlt)
    →Fehler an Anwendung weiterreichen

Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

EF 6.1 – DataAnnotations

- CodeFirst Modell wird intern in das Entity Data Model (EDM) umgewandelt
- Migration werden ausschließlich auf EDM Änderungen erzeugt
- Index Informationen (siehe nächste Feature) spiegeln sich **nicht** wider
- DataAnnotations erlauben das Anbringen von Metadaten an CodeFirst Modellen
  - Diese können zu einem späteren Zeitpunkt ausgewertet werden
  - Z.B. zum Erzeugen von Migrations

EF 6.1 – DataAnnotations

- DataAnnotations per Fluent API hinzufügen

```
protected override void OnModelCreating(DbModelBuilder mBuilder)
{
   mBuilder.Entity<Category>()
     .HasTableAnnotation("MeineAnnotation", "meineDaten");


  mBuilder. Entity<Category>()
    .Property(p => p.Name)
    .HasColumnAnnotation("MeineAnnotation", "meineDaten2");
  …
```

- DataAnnotation anullieren

```
protected override void OnModelCreating(DbModelBuilder mBuilder)
{
   mBuilder.Entity<Category>()
     .HasTableAnnotation("MeineAnnotation", null);
```

EF 6.1 – DataAnnotations

- DataAnnotations per Attribut definieren

```
[AttributeUsage(AttributeTargets.Property |
                AttributeTargets.Class, AllowMultiple = false)]
public class AnnotateMeAttribute : Attribute
{
    public string Data { get; set; }
}
```

- DataAnnotation Attribute auf Column anwenden

```
protected override void OnModelCreating(DbModelBuilder mBuilder)
{
   mBuilder.Conventions.Add(
    new AttributeToColumnAnnotationConvention<AnnotateMeAttribute,
        string>("MyAnnotation",
        (property, attributes) => attributes.Single().Data));
```

EF 6.1 – DataAnnotations

- DataAnnotations werden in Migrations ausgewertet
  - AddColumnOperation: neue Spalte mit Annotations Daten
  - AlterColumnOperation: Spalte mit altem und neuen Annotation Daten
  - DropColumnOperation: Spalte mit Annotation Daten wird gelöscht
  - AddTableOperation : Tabelle mit Annotation Daten wird erzeugt
  - AlterTableOperation: Tabelle mit altem und neuen Annotation Daten
  - DropTableOperation: annotierte Tabelle wird gelöscht

EF 6.1 – DataAnnotations

- Beispiel

```
public partial class Second : DbMigration{
  public override void Up() {
     AlterColumn("dbo.Posts", "Title", c => c.String(
        annotations: new Dictionary<string, AnnotationValues>() {
           {  "MyAnnotation",
              new AnnotationValues(oldValue: "old", newValue: "new")
            },
        }));
  }

  public override void Down() {
     AlterColumn("dbo.Posts", "Title", c => c.String(
        annotations: new Dictionary<string, AnnotationValues>() {
           { "MyAnnotation",
             new AnnotationValues(oldValue: "new", newValue: "old")
           },
        }));
  }
}
```

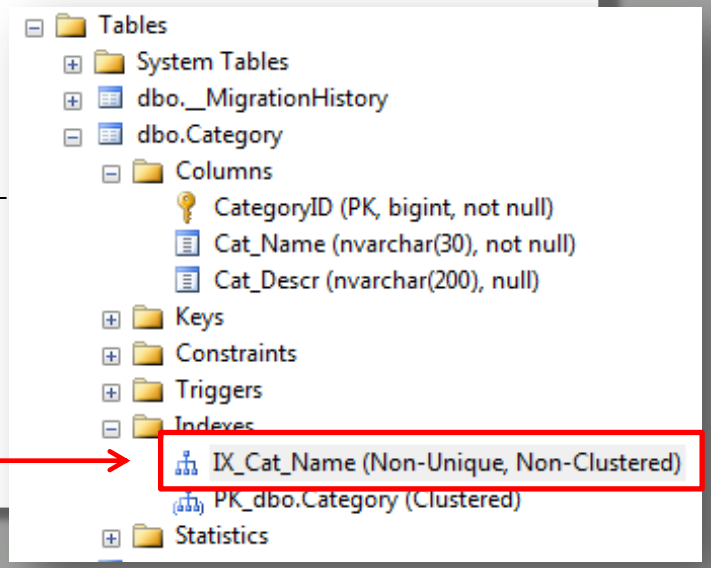
Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

EF 6.1 – IndexAttribute

- Signalisiert bei Code First Entitäten die „Teilnahme“ eines Properties am Index der Tabelle
- Bestandteil der EntityFramework Assembly
- Beispiel

```
using System.ComponentModel.DataAnnotations.Schema;

[Table("Category")]
public class Category
{
    public long CategoryID { get; pri

    [Column("Cat_Name")]
    [StringLength(30)]
    [Required]
    [Index]
    public string Name { get; set; }
```



EF 6.1 – IndexAttribute

- Indexname setzen

```
public class Category
{
    …
    [Index("IndexForName")]
    public string Name { get; set; }
```

- Unique, Clustered

```
public class Category
{
    …
    [Index(IsClustered=true, IsUnique=true)]
    public string Name { get; set; }
```

EF 6.1 – IndexAttribute

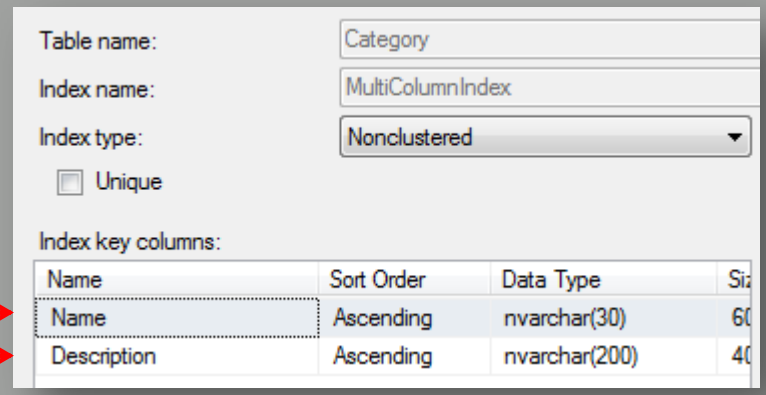
- Index per Data Annotation programmatisch erzeugen

```
public class BugShopContext : DbContext
{
    …
    protected override void OnModelCreating(DbModelBuilder mbuilder)
    {
        var idxAnnotation = new IndexAnnotation(
                    new IndexAttribute("Idx") { IsUnique = true });

        var property = modelBuilder.Properties()
                                   .Where(p => p.Name == "Name");

        property.Configure(p => p.HasColumnAnnotation("Index",
                                                      idxAnnotation));
    }
}
```

EF 6.1 – IndexAttribute

- Multicolumn Index

```
public class Category
{
    …
    [Index("MultiColumn", Order = 1)]
    public string Name { get; set; }

    [Index("MultiColumnIndex", Order = 2)]
    public string Description { get; set; }
```



Entity Framework – Features 6.1

- Interceptoren und Logging
- Behandlung von Transaktions Commit-Fehlern
- Data Annotations
- Index Attribute
- Public Mapping API

EF 6.1 – Public Mapping API

- Bietet die Möglichkeit das per Code First erzeugte EDMX Modell programmatisch zu ändern.

- Beispiel

```
[Table("Category")]
public class Category
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public long CategoryKey { get; private set; }

    [Column("Cat_Name")]
    public string Name { get; set; }


}
```

EF 6.1 – Public Mapping API

- Beispiel

```
public class BugShopContext : DbContext
{
   public virtual IDbSet<Category> Categories { get; set; }
   public virtual IDbSet<SparePart> SpareParts { get; set; }

   protected override void OnModelCreating(DbModelBuilder mB)
   {
      mB.Conventions.Add(new MyConvention());
   }
}
```

EF 6.1 – Public Mapping API

- Beispiel

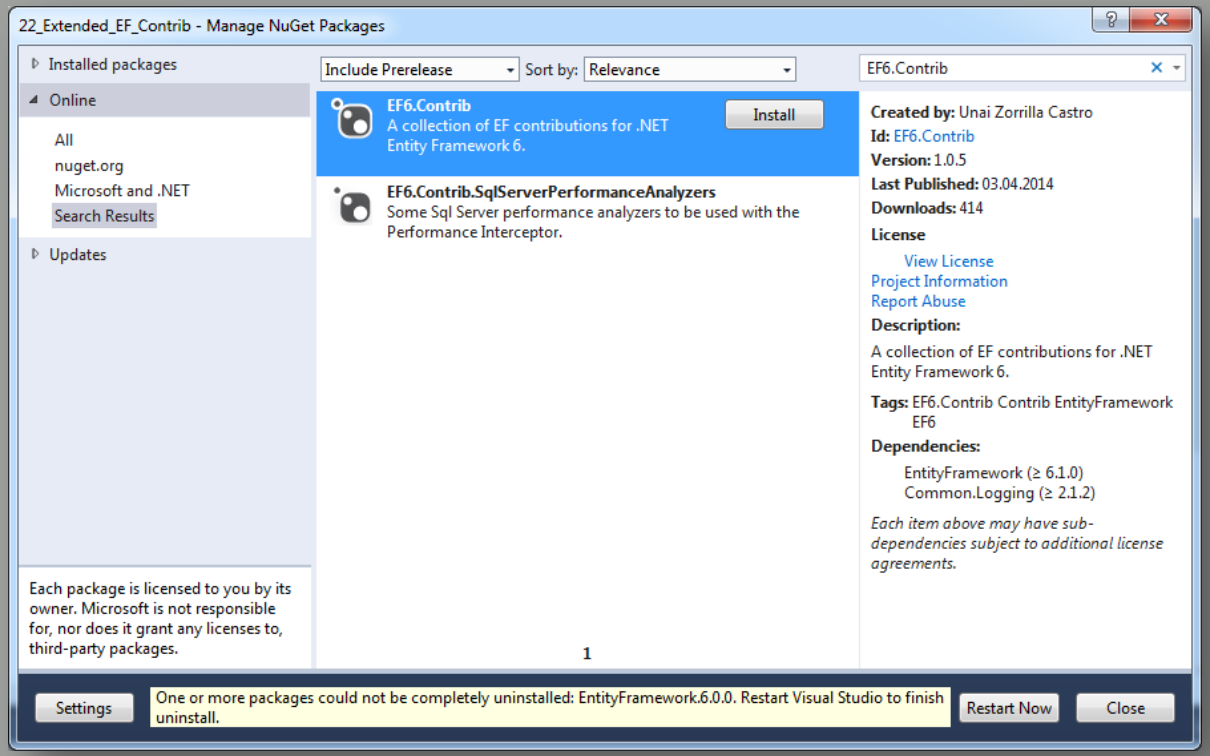
```
public class MyConvention : IStoreModelConvention<EdmModel>
{
    public void Apply(EdmModel item, DbModel model)
    {
        var propertyMapping
            = model.ConceptualToStoreMapping
                .EntitySetMappings.Single()
                .EntityTypeMappings.Single(
                    etm => etm.EntityType.Name == "Category")
                .Fragments.Single()
                .PropertyMappings.Single(
                            pm => pm.Property.Name == "Name");

        var scalarPropertyMapping = (ScalarPropertyMapping)
                                    propertyMapping;

        scalarPropertyMapping.Column.Name = "NeuerName";
    }
}
```

Agenda

- Enitity Framework 6.0 Features
- Enitity Framework 6.1 Features
- Entity Framework 6.x Erweiterungen
- Zusammenfassung

Entity Framework Contrib

- Sammlung von Funktionen, die nicht Bestandteil des EF Frameworks sind.
- https://ef6contrib.codeplex.com/
- Über NuGet
- Lizenz: Apache 2.0



Entity Framework Contrib

- ConfigurationRegistrar
  - AddFromThisAssembly
  - AddFromNamedAssembly
- Pluralization Services
  - Spanish Pluralization Service
- *Migration Operations*
  - *New migrations operations*
- *Initializers*
  - *New Initializers*
- Analyzers
  - Analyzers

Entity Framework Contrib

- ConfigurationRegistrar stellt zwei Extensions-Methoden bereit
  - AddFromThisAssembly

```
public class BugShopContext : DbContext
{
    public DbSet<Category> Customers { get; set; }

    protected override void OnModelCreating(
                         DbModelBuilder modelBuilder)
    {
      modelBuilder.Configurations.AddFromThisAssembly();
    }
}
```

  - AddFromNamedAssembly

```
…
modelBuilder.Configurations.AddFromNamedAssembly("Name");
…
```

Entity Framework Contrib

- Spanish Pluralization Service

```
public class MeineEfKonfiguration : DbConfiguration
{
    public MeineEfKonfiguration()
    {
        SetPluralizationService(new
                        SpanishPluralizationService());
    }
}
```

Entity Framework Contrib

- Analyzers
  - **ExecutionTimePerformanceAnalyzer**
    - Prüft, ob SQL Aufführungszeit größer einem Schwellwert
  - **UnparametrizedWhereClausesPerformanceAnalyzer**
    - Prüft ,ob WHERE-Klausel unparametrisiert ist
  - **TopSlowQueriesPerformanceAnalyzer**
    - Prüft ,ob query zu den „Top slow“ Abfragen des SQL-Servers zählt

Entity Framework Contrib

- Analyzer Beispiel

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure.DependencyResolution;
using System.Data.Entity.Infrastructure.Interception;

class BugshopDbConfig : DbConfiguration
{
    public BugshopDbConfig()
    {
        AddDependencyResolver(new AnalyzerResolver());

        AddInterceptor(new PerformanceInterceptor(
            msg =>
                {
                    Console.WriteLine("Perf: {0}", msg.Message );
                }));
    }
}
```

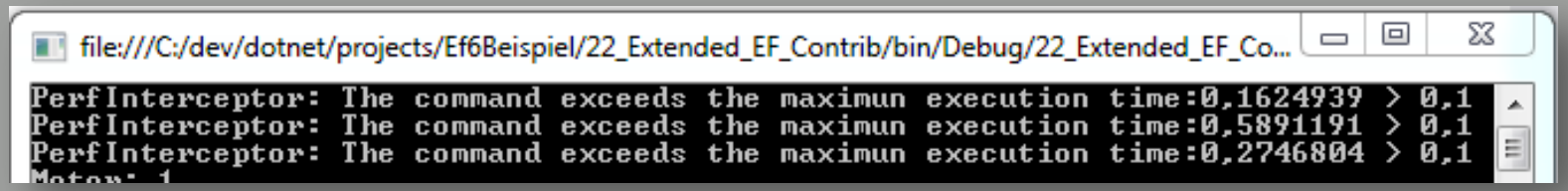
Entity Framework Contrib

- Analyzer Beispiel

```
private class AnalyzerResolver: IDbDependencyResolver
{
   private IList<IPerformanceAnalyzer> analyzers;

   public AnalyzerResolver ()
   {
      analyzers = new List<IPerformanceAnalyzer>() {
          new ExecutionTimePerformanceAnalyzer(
                   TimeSpan.FromMilliseconds(5)) };
   }

   public IEnumerable<object> GetServices(Type typ,
                                          object key)
   {
      if (typeof(IPerformanceAnalyzer).IsAssignableFrom(typ))
      {
         return analyzers;
      }
      return Enumerable.Empty<object>();
   } …
```

Entity Framework Contrib

- Analyzer Beispiel

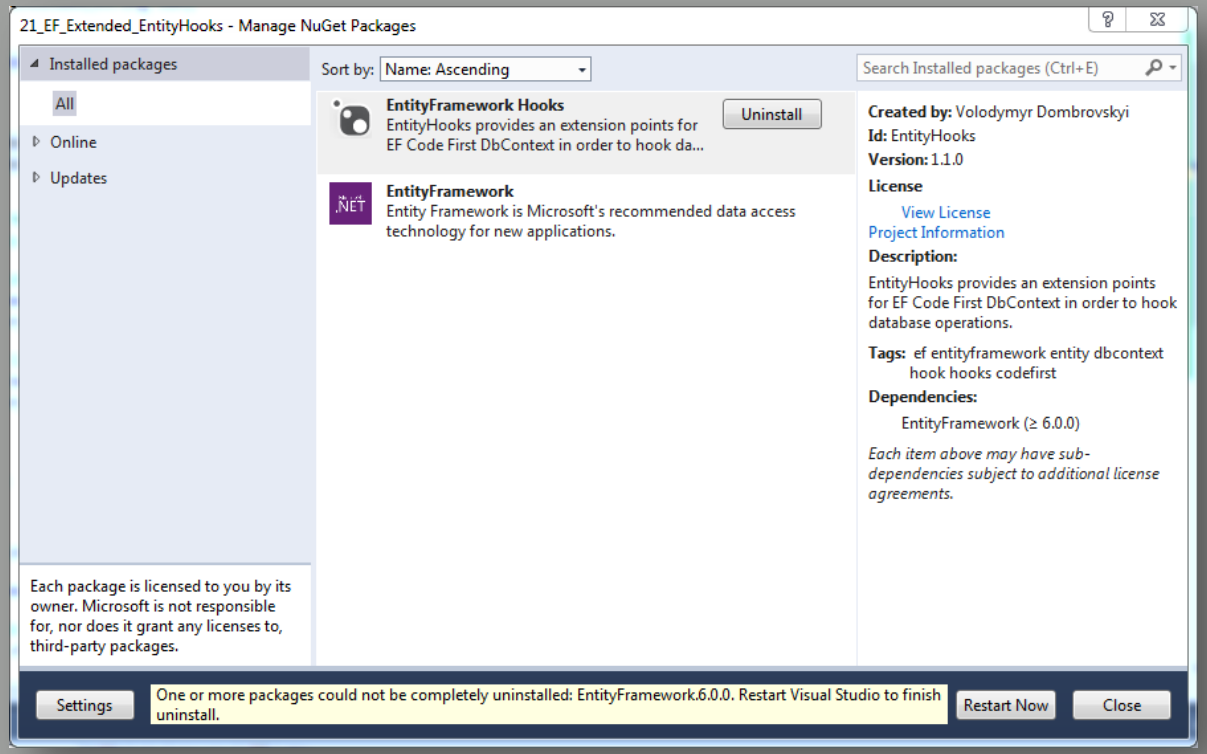


- Weitere Interceptoren

```
public AnalyzerResolver ()
{
    analyzers = new List<IPerformanceAnalyzer>()
    {
        new ExecutionTimePerformanceAnalyzer(
                                TimeSpan.FromMilliseconds(100)),
        new UnparametrizedWhereClausesPerformanceAnalyzer(),
        new UnparametrizedSkipTakeValuesPerformanceAnalyzer(),
        new TopSlowQueriesPerformanceAnalyzer(5)
  };
}
```

EntityHooks

- Einfache Entity Interceptoren für Entity Framework 6.0.0 aufwärts
- http://dombrovsky.github.io/EntityHooks/
- Version 1.1.0 ist verfügbar
- Über NuGet
- Lizenz: MIT



EntityHooks

- Beispiel

```
using System.Data.Entity.Hooks.Fluent;

namespace EntityHooksBeipiel
{
    class Program
    {
        static void Main(string[] args)
        {
            string con = @"Data Source=…";
            using (BugShopContext ctx =
                        new BugShopContext(con))
            {
```

EntityHooks

- OnSave Hook
  - Wird vor dem Speichern der Entität aufgerufen

- Beispiel

```
…
using (BugShopContext ctx = new BugShopContext(con))
{
    ctx.CreateHook()
        .OnSave<Category>()
        .When(EntityState.Added)
        .Do(c =>
            Console.WriteLine("Neue Entität: {0}",
                              c.Name)
          );
```

EntityHooks

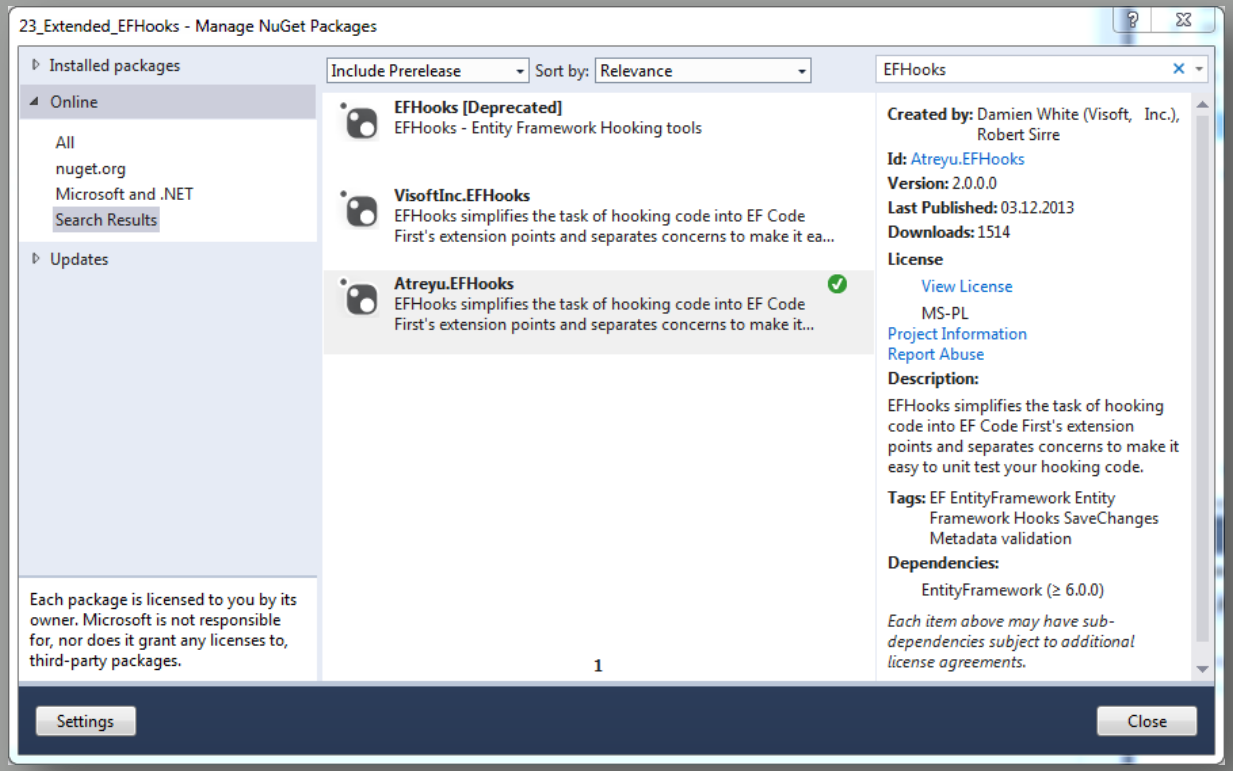
- OnLoad Hook
  - Wird nach dem Laden der Entität aufgerufen

- Beispiel

```
…
using (BugShopContext ctx = new BugShopContext(con))
{
    ctx.CreateHook()
        .OnLoad<Category>()
        .Do(c =>
            Console.WriteLine("Neue Entität: {0}",
                              c.Name)
          );
```

EFHooks

- Einfache Entity Interceptoren für Entity Framework 6.0.0 aufwärts
- https://github.com/Atrejoe/EFHooks
- Version 2.0.0 ist verfügbar
- Über NuGet
- Lizenz: MS-PL
- Vorsicht: Versionsvielfalt



EFHooks

- Bietet streng getypte Hooks
  - PreInsertHook
  - PostInsertHook
  - PreDeleteHook
  - PostDeleteHook
  - PreUpdateHook
  - PostUpdateHook
  - PreActionHook
  - PostActionHook
- Beispiel

```
…
using (BugShopContext ctx = new BugShopContext(con))
{
```

EFHooks

- Eigene DbContext Klasse muss von EFHook.HookedDbContext abgeleitet sein

- Beispiel

```
class BugShopContext : HookedDbContext
{
    static BugShopContext()
    {
      Database.SetInitializer(new
         DropCreateDatabaseIfModelChanges<BugShopContext>());
    }

    public BugShopContext(string nameOrConnectionString)
           : base(nameOrConnectionString)
    {
        this.RegisterHook(new CategoryPreInsertHook());
        this.RegisterHook(new CategoryPostInsertHook());
    }
    …
```

EFHooks

- Eigene Hook Klasse muss von vordefinierten (abstrakten) Hook Klassen abgeleitet sein

- Beispiel

```
public class CategoryPreInsertHook : PreInsertHook<Category>
{
     public override void Hook(Category entity,
                               HookEntityMetadata metadata)
     {
          // hier kommt die „hook“ Logik
     }
 }
```

- Es gibt keine Hooks für geladene Entitäten…

EFHooks (extended)

- IPostLoadHook implementieren

```
namespace EFHooks
{
    public interface IPostLoadHook : IHook
    {}
}
```

- Abstrakte Implementierung

```
public abstract class PostLoadHook<TEntity> : IPostLoadHook
{
    public void HookObject(object e, HookEntityMetadata metadata)
    {
        if (typeof(TEntity).IsAssignableFrom(e.GetType()))
        {
            Hook((TEntity)entity, metadata);
        }
    }
    …
}
```

EFHooks (extended)

- HookDbContext Konstruktoren erweitern, z.B.:

```
public abstract class HookedDbContext : DbContext
{
    …
    protected IList<IPostLoadHook> postLoadHooks;
    private readonly ObjectContext objectContext;

    public HookedDbContext(string nameOrConnectionString)
            : base(nameOrConnectionString)
    {
        PreHooks = new List<IPreActionHook>();
        PostHooks = new List<IPostActionHook>();

        postLoadHooks = new List<IPostLoadHook>();

        objCtx = ((IObjectContextAdapter) this).ObjectContext;
        objCtx.ObjectMaterialized += ObjectMaterialized;
    }
    …
}
```

EFHooks (extended)

- HookDbContext ObjectMaterialized implementieren

```
private void ObjectMaterialized(object sender,
                                ObjectMaterializedEventArgs e)
{
   var md = new HookEntityMetadata(EntityState.Unchanged,
                                    this);

   foreach (var postLoadHook in postLoadHooks)
   {
     postLoadHook.HookObject(e.Entity, metadata);
   }
}
```

EFHooks (extended)

- PostLoadHook verwenden

```
public class LoggingPostLoadHook<TEntity> : PostLoadHook<TEntity>
{
   public override void Hook(TEntity e,
                             HookEntityMetadata metadata)
   {
      Console.WriteLine("PostLoad Entity: {0}", e.ToString());
   }
}
```

```
class BugShopContext : HookedDbContext
{
   public BugShopContext(string conString) : base(conString)
   {
     …
     this.RegisterHook(new LoggingPostLoadHook<Category>());
     this.RegisterHook(new LoggingPostLoadHook<SparePart>());
   }
}
```

Second Level Cache for EF 6.1

- Second Level Caching existiert für EF5 und davor
- Second Level Caching ist für EF6 auf der „Wunschliste“

  - Platz 5

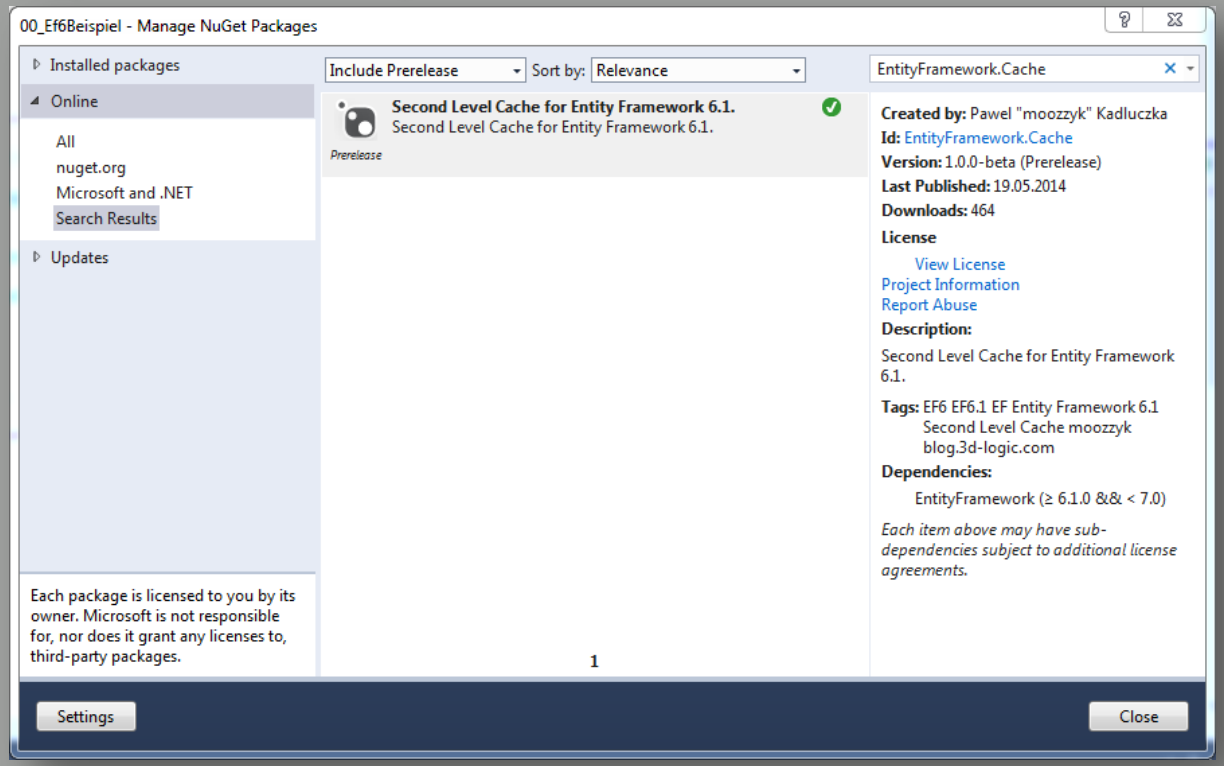
- Aber Diskussion zu diesem *Feature Suggestion* ist „eingeschlafen“
- Laut Rowan Miller wird nach dem Release 6.1 an der Portierung des EF Wrapper Providers für EF 6 gearbeitet

Second Level Cache for EF 6.1

- Das Projekt Second Level Caching for Entity Framework 6.1 versucht diese Lücke zu schließen
- „One man show“
- Ist aktuell eine Beta Version !
- Release 28.07.2014 (beta 2)
- Microsoft Public License (Ms-Pl)

Second Level Cache for EF 6.1

- Bereitgestellt über nuget



- Bietet in-memory Query (-Result) Caching

Second Level Cache for EF 6.1

- Beispiel
  - Default Caching Strategie und InMemoryCache

```
public class Configuration : DbConfiguration
{
  public Configuration()
  {
    var transactionHandler =
              new CacheTransactionHandler(new InMmemoryCache());

    AddInterceptor(transactionHandler);

    var cachingPolicy = new CachingPolicy();

    Loaded +=
      (sender, args) =>
         args.ReplaceService<DbProviderServices(
        (s, _) => new CachingProviderServices(s, transactionHandler,
          cachingPolicy));
  }
}
```

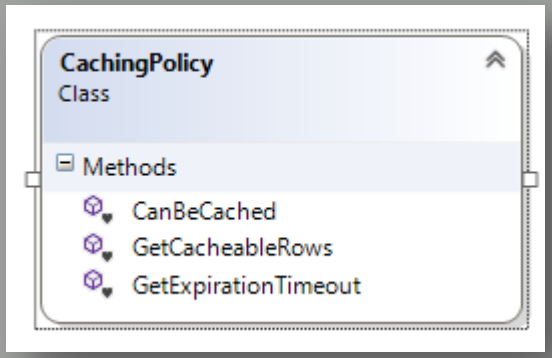
Second Level Cache for EF 6.1

- Beispiel
  - Default Caching Strategie und InMemoryCache

```
public class Configuration : DbConfiguration
{
  public Configuration()
  {
    var transactionHandler =
              new CacheTransactionHandler(new InMmemoryCache());

    AddInterceptor(transactionHandler);

    var cachingPolicy = new CachingPolicy();

    Loaded +=
      (sender, args) =>
         args.ReplaceService<DbProviderServices(
        (s, _) => new CachingProviderServices(s, transactionHandler,
          cachingPolicy));
  }
}
```

Second Level Cache for EF 6.1

- Was bedeutet Default Caching Strategie?



- Alle Query Ergebnisse werden gecached, unabhängig von deren Größe

- “Expiration” ist auf Maximum gesetzt.
  - D.h. die Daten werden so lange gecached bis ein EntitySet, das bei der Query genutzt wurde, verändert wird.

Second Level Cache for EF 6.1 (extended)

- Default Caching Strategie ersetzen
- Es sollen nur Queries von Entitäten im Second-Level Cache platziert werden, die als Cacheable markiert sind (Analogie zu JPA 2)

- Implementierungsskizze

```
[AttributeUsage(AttributeTargets.Class)]
public class CacheableAttribute : Attribute
{
}
```

```
[Cacheable]
public class Category
{
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public long CategoryID { get; private set; }

   …
}
```

Second Level Cache for EF 6.1 (extended)

- Implementierungsskizze

```
public class BugShopContext : DbContext
{
    public IDbSet<Category> Categories { get; set; }
    public IDbSet<SparePart> SpareParts { get; set; }

    /// <summary>
    /// Konfig basiert auf dem Blog
    /// http://romiller.com/2014/04/08/ef6-1-mapping-between-types-tables/
    /// </summary>
    /// <param name="nameOrConnectionString"></param>
    public BugShopContext(string conString): base(conString)
    {
        FindCacheableEntityClasses();
    }

    private void FindCacheableEntityClasses()
    {
        // nächste Folie
    }
}
```

Second Level Cache for EF 6.1 (extended)

- Implementierungsskizze

```
private void FindCacheableEntityClasses()
{
   var metadata = ((IObjectContextAdapter)
                     this).ObjectContext.MetadataWorkspace;

   var objectItemCollection = ((ObjectItemCollection)
             metadata.GetItemCollection(DataSpace.OSpace));
   var entityTypes =
           metadata.GetItems<EntityType>(DataSpace.OSpace);
   foreach (var entityType in entityTypes)
   {
      var clrType = objectItemCollection.GetClrType(entityType);
      if (Attribute.GetCustomAttributes(clrType)
         .SingleOrDefault(a => a is CacheableAttribute) != null) {
         GetAndAddEntitySetToRegistry(clrType);
      }
   }
}
```

Second Level Cache for EF 6.1 (extended)

- Implementierungsskizze

```
class DemoCachingPolicy : CachingPolicy
{
   protected override bool CanBeCached(
           ReadOnlyCollection<EntitySetBase> affectedEntitySets,
           string sql,
           IEnumerable<KeyValuePair<string, object>> parameters)
   {
      foreach (EntitySetBase entitySetBase in affectedEntitySets)
      {
        if (!CachableEntitiesRegistry.Contains(entitySetBase))
        {
           return false;
        }
      }
      return true;
   }
}
```

Second Level Cache for EF 6.1 (extended)

- Implementierungsskizze

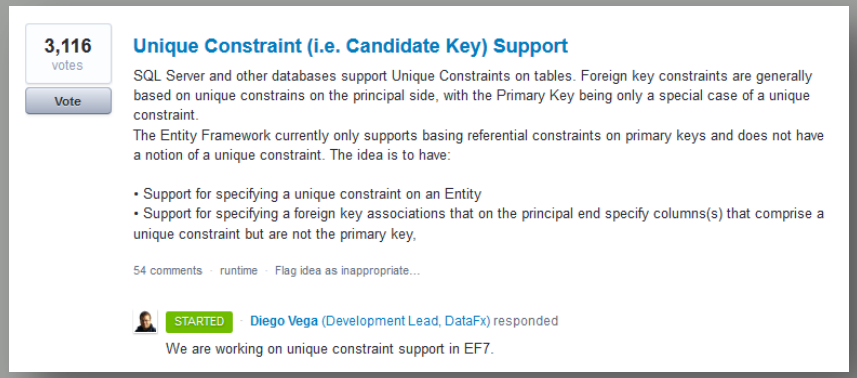
```
public void GetAndAddEntitySetToRegistry(Type entityType)
{
   var metadata = ((IObjectContextAdapter)
                  this).ObjectContext.MetadataWorkspace;

   var storageESet =
      metadata.GetItems<EntityContainer>(DataSpace.SSpace)
              .Single()
              .EntitySets
              .Single(s => s.ElementType.Name == entityType.Name);

   CachableEntitiesRegistry.AddCacheableEntitySet(storageESet);
}
```

Agenda

- Enitity Framework 6.0 Features
- Enitity Framework 6.1 Features
- Entity Framework 6.x Erweiterungen
- Zusammenfassung

Zusammenfassung

- Entity Framework 6 und 6.1 bringen wichtige und notwendige Neuerungen
- Erweiterungen
  - versuchen wichtige Features nachzubilden (Second Level Caching)
  - Features und “Reifegrad” der Erweiterungen befinden sich im Aufbau
- *Was fehlt mir*



Fragen

Viel Spaß mit dem Entity Framework

thomas.haug@mathema.de
@tshaug
www.sharpmetrics.net

Durchhänger?

Eine kleine Stärkung gibt es jetzt am MATHEMA-Stand!