**FIND THE YELLOW HIGHLIGHTED TEXT IN THIS FILE AND MODIFY THE CONTEXT TEMPLATE ACCORDINGLY**

<#@ template language="C#" debug="false" hostspecific="true"#>

<#@ include file="EF6.Utility.CS.ttinclude"#><#@

output extension=".cs"#><#

const string inputFile = @"WellMaster.edmx";

var textTransform = DynamicTextTransformation.Create(this);

var code = new CodeGenerationTools(this);

var ef = new MetadataTools(this);

var typeMapper = new TypeMapper(code, ef, textTransform.Errors);

var loader = new EdmMetadataLoader(textTransform.Host, textTransform.Errors);

var itemCollection = loader.CreateEdmItemCollection(inputFile);

var modelNamespace = loader.GetModelNamespace(inputFile);

var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);

var container = itemCollection.OfType<EntityContainer>().FirstOrDefault();

if (container == null)

{

return string.Empty;

}

#>

//------------------------------------------------------------------------------

// <auto-generated>

// <#=CodeGenerationTools.GetResourceString("Template\_GeneratedCodeCommentLine1")#>

//

// <#=CodeGenerationTools.GetResourceString("Template\_GeneratedCodeCommentLine2")#>

// <#=CodeGenerationTools.GetResourceString("Template\_GeneratedCodeCommentLine3")#>

// </auto-generated>

//------------------------------------------------------------------------------

<#

var codeNamespace = code.VsNamespaceSuggestion();

if (!String.IsNullOrEmpty(codeNamespace))

{

#>

namespace <#=code.EscapeNamespace(codeNamespace)#>

{

<#

PushIndent(" ");

}

#>

using System;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

<#@ include file="C:\DEV-MAIN\Main\Libs\Org.GS\Other\EFHelper.ttinclude"#>

<#

if (container.FunctionImports.Any())

{

#>

using System.Data.Entity.Core.Objects;

using System.Linq;

<#

}

#>

[DbMap(DbElement.EntitySet, "", "", "")]

<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext

{

public <#=code.Escape(container)#>(string connectionStringName)

: base(connectionStringName)

{

<#

if (!loader.IsLazyLoadingEnabled(container))

{

#>

this.Configuration.LazyLoadingEnabled = false;

<#

}

foreach (var entitySet in container.BaseEntitySets.OfType<EntitySet>())

{

// Note: the DbSet members are defined below such that the getter and

// setter always have the same accessibility as the DbSet definition

if (Accessibility.ForReadOnlyProperty(entitySet) != "public")

{

#>

<#=codeStringGenerator.DbSetInitializer(entitySet)#>

<#

}

}

#>

}

public <#=code.Escape(container)#>(ConfigDbSpec configDbSpec)

: base(configDbSpec.ConnectionString)

{}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

<#

foreach (var entitySet in container.BaseEntitySets.OfType<EntitySet>())

{

#>

<#=codeStringGenerator.DbSet(entitySet)#>

<#

}

foreach (var edmFunction in container.FunctionImports)

{

WriteFunctionImport(typeMapper, codeStringGenerator, edmFunction, modelNamespace, includeMergeOption: false);

}

#>

}

<#

if (!String.IsNullOrEmpty(codeNamespace))

{

PopIndent();

#>

}

<#

}

#>

<#+

private void WriteFunctionImport(TypeMapper typeMapper, CodeStringGenerator codeStringGenerator, EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

{

if (typeMapper.IsComposable(edmFunction))

{

#>

[DbFunction("<#=edmFunction.NamespaceName#>", "<#=edmFunction.Name#>")]

<#=codeStringGenerator.ComposableFunctionMethod(edmFunction, modelNamespace)#>

{

<#+

codeStringGenerator.WriteFunctionParameters(edmFunction, WriteFunctionParameter);

#>

<#=codeStringGenerator.ComposableCreateQuery(edmFunction, modelNamespace)#>

}

<#+

}

else

{

#>

<#=codeStringGenerator.FunctionMethod(edmFunction, modelNamespace, includeMergeOption)#>

{

<#+

codeStringGenerator.WriteFunctionParameters(edmFunction, WriteFunctionParameter);

#>

<#=codeStringGenerator.ExecuteFunction(edmFunction, modelNamespace, includeMergeOption)#>

}

<#+

if (typeMapper.GenerateMergeOptionFunction(edmFunction, includeMergeOption))

{

WriteFunctionImport(typeMapper, codeStringGenerator, edmFunction, modelNamespace, includeMergeOption: true);

}

}

}

public void WriteFunctionParameter(string name, string isNotNull, string notNullInit, string nullInit)

{

#>

var <#=name#> = <#=isNotNull#> ?

<#=notNullInit#> :

<#=nullInit#>;

<#+

}

public const string TemplateId = "CSharp\_DbContext\_Context\_EF6";

public class CodeStringGenerator

{

private readonly CodeGenerationTools \_code;

private readonly TypeMapper \_typeMapper;

private readonly MetadataTools \_ef;

public CodeStringGenerator(CodeGenerationTools code, TypeMapper typeMapper, MetadataTools ef)

{

ArgumentNotNull(code, "code");

ArgumentNotNull(typeMapper, "typeMapper");

ArgumentNotNull(ef, "ef");

\_code = code;

\_typeMapper = typeMapper;

\_ef = ef;

}

public string Property(EdmProperty edmProperty)

{

return string.Format(

CultureInfo.InvariantCulture,

"{0} {1} {2} {{ {3}get; {4}set; }}",

Accessibility.ForProperty(edmProperty),

\_typeMapper.GetTypeName(edmProperty.TypeUsage),

\_code.Escape(edmProperty),

\_code.SpaceAfter(Accessibility.ForGetter(edmProperty)),

\_code.SpaceAfter(Accessibility.ForSetter(edmProperty)));

}

public string NavigationProperty(NavigationProperty navProp)

{

var endType = \_typeMapper.GetTypeName(navProp.ToEndMember.GetEntityType());

return string.Format(

CultureInfo.InvariantCulture,

"{0} {1} {2} {{ {3}get; {4}set; }}",

AccessibilityAndVirtual(Accessibility.ForNavigationProperty(navProp)),

navProp.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many ? ("ICollection<" + endType + ">") : endType,

\_code.Escape(navProp),

\_code.SpaceAfter(Accessibility.ForGetter(navProp)),

\_code.SpaceAfter(Accessibility.ForSetter(navProp)));

}

public string AccessibilityAndVirtual(string accessibility)

{

return accessibility + (accessibility != "private" ? " virtual" : "");

}

public string EntityClassOpening(EntityType entity)

{

return string.Format(

CultureInfo.InvariantCulture,

"{0} {1}partial class {2}{3}",

Accessibility.ForType(entity),

\_code.SpaceAfter(\_code.AbstractOption(entity)),

\_code.Escape(entity),

\_code.StringBefore(" : ", \_typeMapper.GetTypeName(entity.BaseType)));

}

public string EnumOpening(SimpleType enumType)

{

return string.Format(

CultureInfo.InvariantCulture,

"{0} enum {1} : {2}",

Accessibility.ForType(enumType),

\_code.Escape(enumType),

\_code.Escape(\_typeMapper.UnderlyingClrType(enumType)));

}

public void WriteFunctionParameters(EdmFunction edmFunction, Action<string, string, string, string> writeParameter)

{

var parameters = FunctionImportParameter.Create(edmFunction.Parameters, \_code, \_ef);

foreach (var parameter in parameters.Where(p => p.NeedsLocalVariable))

{

var isNotNull = parameter.IsNullableOfT ? parameter.FunctionParameterName + ".HasValue" : parameter.FunctionParameterName + " != null";

var notNullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", " + parameter.FunctionParameterName + ")";

var nullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", typeof(" + TypeMapper.FixNamespaces(parameter.RawClrTypeName) + "))";

writeParameter(parameter.LocalVariableName, isNotNull, notNullInit, nullInit);

}

}

public string ComposableFunctionMethod(EdmFunction edmFunction, string modelNamespace)

{

var parameters = \_typeMapper.GetParameters(edmFunction);

return string.Format(

CultureInfo.InvariantCulture,

"{0} IQueryable<{1}> {2}({3})",

AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),

\_typeMapper.GetTypeName(\_typeMapper.GetReturnType(edmFunction), modelNamespace),

\_code.Escape(edmFunction),

string.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) + " " + p.FunctionParameterName).ToArray()));

}

public string ComposableCreateQuery(EdmFunction edmFunction, string modelNamespace)

{

var parameters = \_typeMapper.GetParameters(edmFunction);

return string.Format(

CultureInfo.InvariantCulture,

"return ((IObjectContextAdapter)this).ObjectContext.CreateQuery<{0}>(\"[{1}].[{2}]({3})\"{4});",

\_typeMapper.GetTypeName(\_typeMapper.GetReturnType(edmFunction), modelNamespace),

edmFunction.NamespaceName,

edmFunction.Name,

string.Join(", ", parameters.Select(p => "@" + p.EsqlParameterName).ToArray()),

\_code.StringBefore(", ", string.Join(", ", parameters.Select(p => p.ExecuteParameterName).ToArray())));

}

public string FunctionMethod(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

{

var parameters = \_typeMapper.GetParameters(edmFunction);

var returnType = \_typeMapper.GetReturnType(edmFunction);

var paramList = String.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) + " " + p.FunctionParameterName).ToArray());

if (includeMergeOption)

{

paramList = \_code.StringAfter(paramList, ", ") + "MergeOption mergeOption";

}

return string.Format(

CultureInfo.InvariantCulture,

"{0} {1} {2}({3})",

AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),

returnType == null ? "int" : "ObjectResult<" + \_typeMapper.GetTypeName(returnType, modelNamespace) + ">",

\_code.Escape(edmFunction),

paramList);

}

public string ExecuteFunction(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

{

var parameters = \_typeMapper.GetParameters(edmFunction);

var returnType = \_typeMapper.GetReturnType(edmFunction);

var callParams = \_code.StringBefore(", ", String.Join(", ", parameters.Select(p => p.ExecuteParameterName).ToArray()));

if (includeMergeOption)

{

callParams = ", mergeOption" + callParams;

}

return string.Format(

CultureInfo.InvariantCulture,

"return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction{0}(\"{1}\"{2});",

returnType == null ? "" : "<" + \_typeMapper.GetTypeName(returnType, modelNamespace) + ">",

edmFunction.Name,

callParams);

}

public string DbSet(EntitySet entitySet)

{

return string.Format(

CultureInfo.InvariantCulture,

"{0} virtual DbSet<{1}> {2} {{ get; set; }}",

Accessibility.ForReadOnlyProperty(entitySet),

\_typeMapper.GetTypeName(entitySet.ElementType),

\_code.Escape(entitySet));

}

public string DbSetInitializer(EntitySet entitySet)

{

return string.Format(

CultureInfo.InvariantCulture,

"{0} = Set<{1}>();",

\_code.Escape(entitySet),

\_typeMapper.GetTypeName(entitySet.ElementType));

}

public string UsingDirectives(bool inHeader, bool includeCollections = true)

{

return inHeader == string.IsNullOrEmpty(\_code.VsNamespaceSuggestion())

? string.Format(

CultureInfo.InvariantCulture,

"{0}using System;{1}" +

"{2}",

inHeader ? Environment.NewLine : "",

includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",

inHeader ? "" : Environment.NewLine)

: "";

}

}

public class TypeMapper

{

private const string ExternalTypeNameAttributeName = @"http://schemas.microsoft.com/ado/2006/04/codegeneration:ExternalTypeName";

private readonly System.Collections.IList \_errors;

private readonly CodeGenerationTools \_code;

private readonly MetadataTools \_ef;

public static string FixNamespaces(string typeName)

{

return typeName.Replace("System.Data.Spatial.", "System.Data.Entity.Spatial.");

}

public TypeMapper(CodeGenerationTools code, MetadataTools ef, System.Collections.IList errors)

{

ArgumentNotNull(code, "code");

ArgumentNotNull(ef, "ef");

ArgumentNotNull(errors, "errors");

\_code = code;

\_ef = ef;

\_errors = errors;

}

public string GetTypeName(TypeUsage typeUsage)

{

return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, \_ef.IsNullable(typeUsage), modelNamespace: null);

}

public string GetTypeName(EdmType edmType)

{

return GetTypeName(edmType, isNullable: null, modelNamespace: null);

}

public string GetTypeName(TypeUsage typeUsage, string modelNamespace)

{

return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, \_ef.IsNullable(typeUsage), modelNamespace);

}

public string GetTypeName(EdmType edmType, string modelNamespace)

{

return GetTypeName(edmType, isNullable: null, modelNamespace: modelNamespace);

}

public string GetTypeName(EdmType edmType, bool? isNullable, string modelNamespace)

{

if (edmType == null)

{

return null;

}

var collectionType = edmType as CollectionType;

if (collectionType != null)

{

return String.Format(CultureInfo.InvariantCulture, "ICollection<{0}>", GetTypeName(collectionType.TypeUsage, modelNamespace));

}

var typeName = \_code.Escape(edmType.MetadataProperties

.Where(p => p.Name == ExternalTypeNameAttributeName)

.Select(p => (string)p.Value)

.FirstOrDefault())

?? (modelNamespace != null && edmType.NamespaceName != modelNamespace ?

\_code.CreateFullName(\_code.EscapeNamespace(edmType.NamespaceName), \_code.Escape(edmType)) :

\_code.Escape(edmType));

if (edmType is StructuralType)

{

return typeName;

}

if (edmType is SimpleType)

{

var clrType = UnderlyingClrType(edmType);

if (!IsEnumType(edmType))

{

typeName = \_code.Escape(clrType);

}

typeName = FixNamespaces(typeName);

return clrType.IsValueType && isNullable == true ?

String.Format(CultureInfo.InvariantCulture, "Nullable<{0}>", typeName) :

typeName;

}

throw new ArgumentException("edmType");

}

public Type UnderlyingClrType(EdmType edmType)

{

ArgumentNotNull(edmType, "edmType");

var primitiveType = edmType as PrimitiveType;

if (primitiveType != null)

{

return primitiveType.ClrEquivalentType;

}

if (IsEnumType(edmType))

{

return GetEnumUnderlyingType(edmType).ClrEquivalentType;

}

return typeof(object);

}

public object GetEnumMemberValue(MetadataItem enumMember)

{

ArgumentNotNull(enumMember, "enumMember");

var valueProperty = enumMember.GetType().GetProperty("Value");

return valueProperty == null ? null : valueProperty.GetValue(enumMember, null);

}

public string GetEnumMemberName(MetadataItem enumMember)

{

ArgumentNotNull(enumMember, "enumMember");

var nameProperty = enumMember.GetType().GetProperty("Name");

return nameProperty == null ? null : (string)nameProperty.GetValue(enumMember, null);

}

public System.Collections.IEnumerable GetEnumMembers(EdmType enumType)

{

ArgumentNotNull(enumType, "enumType");

var membersProperty = enumType.GetType().GetProperty("Members");

return membersProperty != null

? (System.Collections.IEnumerable)membersProperty.GetValue(enumType, null)

: Enumerable.Empty<MetadataItem>();

}

public bool EnumIsFlags(EdmType enumType)

{

ArgumentNotNull(enumType, "enumType");

var isFlagsProperty = enumType.GetType().GetProperty("IsFlags");

return isFlagsProperty != null && (bool)isFlagsProperty.GetValue(enumType, null);

}

public bool IsEnumType(GlobalItem edmType)

{

ArgumentNotNull(edmType, "edmType");

return edmType.GetType().Name == "EnumType";

}

public PrimitiveType GetEnumUnderlyingType(EdmType enumType)

{

ArgumentNotNull(enumType, "enumType");

return (PrimitiveType)enumType.GetType().GetProperty("UnderlyingType").GetValue(enumType, null);

}

public string CreateLiteral(object value)

{

if (value == null || value.GetType() != typeof(TimeSpan))

{

return \_code.CreateLiteral(value);

}

return string.Format(CultureInfo.InvariantCulture, "new TimeSpan({0})", ((TimeSpan)value).Ticks);

}

public bool VerifyCaseInsensitiveTypeUniqueness(IEnumerable<string> types, string sourceFile)

{

ArgumentNotNull(types, "types");

ArgumentNotNull(sourceFile, "sourceFile");

var hash = new HashSet<string>(StringComparer.InvariantCultureIgnoreCase);

if (types.Any(item => !hash.Add(item)))

{

\_errors.Add(

new CompilerError(sourceFile, -1, -1, "6023",

String.Format(CultureInfo.CurrentCulture, CodeGenerationTools.GetResourceString("Template\_CaseInsensitiveTypeConflict"))));

return false;

}

return true;

}

public IEnumerable<SimpleType> GetEnumItemsToGenerate(IEnumerable<GlobalItem> itemCollection)

{

return GetItemsToGenerate<SimpleType>(itemCollection)

.Where(e => IsEnumType(e));

}

public IEnumerable<T> GetItemsToGenerate<T>(IEnumerable<GlobalItem> itemCollection) where T: EdmType

{

return itemCollection

.OfType<T>()

.Where(i => !i.MetadataProperties.Any(p => p.Name == ExternalTypeNameAttributeName))

.OrderBy(i => i.Name);

}

public IEnumerable<string> GetAllGlobalItems(IEnumerable<GlobalItem> itemCollection)

{

return itemCollection

.Where(i => i is EntityType || i is ComplexType || i is EntityContainer || IsEnumType(i))

.Select(g => GetGlobalItemName(g));

}

public string GetGlobalItemName(GlobalItem item)

{

if (item is EdmType)

{

return ((EdmType)item).Name;

}

else

{

return ((EntityContainer)item).Name;

}

}

public IEnumerable<EdmProperty> GetSimpleProperties(EntityType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);

}

public IEnumerable<EdmProperty> GetSimpleProperties(ComplexType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);

}

public IEnumerable<EdmProperty> GetComplexProperties(EntityType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);

}

public IEnumerable<EdmProperty> GetComplexProperties(ComplexType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);

}

public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(EntityType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type && p.DefaultValue != null);

}

public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(ComplexType type)

{

return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type && p.DefaultValue != null);

}

public IEnumerable<NavigationProperty> GetNavigationProperties(EntityType type)

{

return type.NavigationProperties.Where(np => np.DeclaringType == type);

}

public IEnumerable<NavigationProperty> GetCollectionNavigationProperties(EntityType type)

{

return type.NavigationProperties.Where(np => np.DeclaringType == type && np.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many);

}

public FunctionParameter GetReturnParameter(EdmFunction edmFunction)

{

ArgumentNotNull(edmFunction, "edmFunction");

var returnParamsProperty = edmFunction.GetType().GetProperty("ReturnParameters");

return returnParamsProperty == null

? edmFunction.ReturnParameter

: ((IEnumerable<FunctionParameter>)returnParamsProperty.GetValue(edmFunction, null)).FirstOrDefault();

}

public bool IsComposable(EdmFunction edmFunction)

{

ArgumentNotNull(edmFunction, "edmFunction");

var isComposableProperty = edmFunction.GetType().GetProperty("IsComposableAttribute");

return isComposableProperty != null && (bool)isComposableProperty.GetValue(edmFunction, null);

}

public IEnumerable<FunctionImportParameter> GetParameters(EdmFunction edmFunction)

{

return FunctionImportParameter.Create(edmFunction.Parameters, \_code, \_ef);

}

public TypeUsage GetReturnType(EdmFunction edmFunction)

{

var returnParam = GetReturnParameter(edmFunction);

return returnParam == null ? null : \_ef.GetElementType(returnParam.TypeUsage);

}

public bool GenerateMergeOptionFunction(EdmFunction edmFunction, bool includeMergeOption)

{

var returnType = GetReturnType(edmFunction);

return !includeMergeOption && returnType != null && returnType.EdmType.BuiltInTypeKind == BuiltInTypeKind.EntityType;

}

}

public static void ArgumentNotNull<T>(T arg, string name) where T : class

{

if (arg == null)

{

throw new ArgumentNullException(name);

}

}

#>