This PDF contains the contents of folders and files from the directory 'Monitoring.RabbitMQAlerter' and its subdirectories.

App.config (Monitoring.RabbitMQAlerter\App.config):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <appSettings>
    <add key="StartHour" value="0" />
    <add key="EndHour" value="23" />

    <add key="smtp1Server" value="dvsmtp.plex.com" />
    <add key="smtp1Port" value="25" />
    <add key="smtp1EnableSSL" value="false" />

    <add key="errorEmail" value="obose.uwadiale@rockwellautomation.com" />
    <add key="islocal" value="true" />
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
  </startup>
  <connectionStrings>
    <add name="MetaAzureEntities"
```

connectionString="metadata=res://*/MetaAzure.csdl|res://*/MetaAzure.ssdl|res://*/MetaAzure.msl;provider=System.Data.SqlClient;provider connection string="data source=localhost;initial catalog=AppUserSettings;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"" providerName="System.Data.EntityClient" />
  </connectionStrings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="mssqllocaldb" />
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Threading.Tasks.Extensions" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-4.2.0.1" newVersion="4.2.0.1" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>

```
        <assemblyIdentity name="System.Memory" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-4.0.1.2" newVersion="4.0.1.2" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Threading.Channels" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-7.0.0.0" newVersion="7.0.0.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

DbConnection.cs (Monitoring.RabbitMQAlerter\DbConnection.cs):

```
using Monitoring.ServiceRestarter;

namespace Monitoring.RabbitMQAlerter
{
    /// <summary>
    /// Provides methods to handle the various database connections
    /// </summary>
    /// <remarks>
    /// This is here for future expansion where the connection string might be set programmatically
    /// </remarks>
    public static class DbConnection
    {
        public static string GetMetaAzureConnectionString()
        {
```

```
            var s = Utils.GetInfrastructureConnectionString("auth_monitoring", "AppUserSettings");
            return s;
        }
    }

    public class MetaAzureDbContext : MetaAzureEntities
    {
        public MetaAzureDbContext()
        {
            Database.Connection.ConnectionString = DbConnection.GetMetaAzureConnectionString();
        }
    }
}
```

MetaAzure.Context.cs (Monitoring.RabbitMQAlerter\MetaAzure.Context.cs):

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Monitoring.RabbitMQAlerter
{
```

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

    [DbConfigurationType(typeof(MyDbConfiguration))] // Use the custom DbConfiguration
    public partial class MetaAzureEntities : DbContext
    {
        public MetaAzureEntities()
            : base("name=MetaAzureEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<MetaRabbitQueue> MetaRabbitQueues { get; set; }
        public virtual DbSet<MetaRabbitServer> MetaRabbitServers { get; set; }
        public virtual DbSet<MetaRabbitServer2Queue> MetaRabbitServer2Queue { get; set; }
    }
}
```

MetaAzure.Context.tt (Monitoring.RabbitMQAlerter\MetaAzure.Context.tt):

```
<#@ template language="C#" debug="false" hostspecific="true"#>
<#@ include file="EF6.Utility.CS.ttinclude"#><#@
```

```
 output extension=".cs"#><#


const string inputFile = @"MetaAzure.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;
<#
if (container.FunctionImports.Any())
{
#>
using System.Data.Entity.Core.Objects;
using System.Linq;
<#
}
#>
```

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext
{
    public <#=code.Escape(container)#>()
        : base("name=<#=container.Name#>")
    {
<#
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)#>
<#
    }
}
#>
```

```
    }

    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);
    }
}
#>
```

MetaAzure.cs (Monitoring.RabbitMQAlerter\MetaAzure.cs):

```
//------------------------------------------------------------------------------
// <auto-generated>
```

```
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

MetaAzure.Designer.cs (Monitoring.RabbitMQAlerter\MetaAzure.Designer.cs):

```
// T4 code generation is enabled for model 'C:\Projects\Monitoring.ETL.Domain\Monitoring.RabbitMQAlerter\MetaAzure.edmx'.
// To enable legacy code generation, change the value of the 'Code Generation Strategy' designer
// property to 'Legacy ObjectContext'. This property is available in the Properties Window when the model
// is open in the designer.

// If no context and entity classes have been generated, it may be because you created an empty model but
// have not yet chosen which version of Entity Framework to use. To generate a context class and entity
// classes for your model, open the model in the designer, right-click on the designer surface, and
// select 'Update Model from Database...', 'Generate Database from Model...', or 'Add Code Generation
// Item...'.
```

MetaAzure.edmx (Monitoring.RabbitMQAlerter\MetaAzure.edmx):

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
```

```
  <edmx:StorageModels>
    <Schema Namespace="MetaAzureModel.Store" Provider="System.Data.SqlClient" ProviderManifestToken="2012" Alias="Self"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns:customannotation="http://schemas.microsoft.com/ado/2013/11/edm/customannotation"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">
      <EntityType Name="MetaRabbitQueue">
        <Key>
          <PropertyRef Name="RabbitQueueId" />
        </Key>
        <Property Name="RabbitQueueId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
        <Property Name="RabbitQueueName" Type="varchar" MaxLength="100" />
      </EntityType>
      <EntityType Name="MetaRabbitServer">
        <Key>
          <PropertyRef Name="RabbitServerId" />
        </Key>
        <Property Name="RabbitServerId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
        <Property Name="Environment" Type="varchar" MaxLength="50" Nullable="false" />
        <Property Name="FriendlyName" Type="varchar" MaxLength="50" Nullable="false" />
        <Property Name="WebHostName" Type="varchar" MaxLength="200" Nullable="false" />
        <Property Name="Username" Type="varchar" MaxLength="50" Nullable="false" />
        <Property Name="ServerHostName" Type="varchar" MaxLength="200" />
        <Property Name="ServerHostPortNumber" Type="int" Nullable="false" />
        <Property Name="SortOrder" Type="int" Nullable="false" />
      </EntityType>
      <EntityType Name="MetaRabbitServer2Queue">
```

```
    <Key>
      <PropertyRef Name="RabbitId" />
    </Key>
    <Property Name="RabbitId" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />
    <Property Name="RabbitQueueId" Type="int" Nullable="false" />
    <Property Name="RabbitServerId" Type="int" Nullable="false" />
    <Property Name="MaxThreshold" Type="int" Nullable="false" />
    <Property Name="IsAlert" Type="bit" Nullable="false" />
  </EntityType>
  <Association Name="FK_MetaRabbitServer2Queue_MetaRabbitQueue">
    <End Role="MetaRabbitQueue" Type="Self.MetaRabbitQueue" Multiplicity="1" />
    <End Role="MetaRabbitServer2Queue" Type="Self.MetaRabbitServer2Queue" Multiplicity="*" />
    <ReferentialConstraint>
      <Principal Role="MetaRabbitQueue">
        <PropertyRef Name="RabbitQueueId" />
      </Principal>
      <Dependent Role="MetaRabbitServer2Queue">
        <PropertyRef Name="RabbitQueueId" />
      </Dependent>
    </ReferentialConstraint>
  </Association>
  <Association Name="FK_MetaRabbitServer2Queue_MetaRabbitServer">
    <End Role="MetaRabbitServer" Type="Self.MetaRabbitServer" Multiplicity="1" />
    <End Role="MetaRabbitServer2Queue" Type="Self.MetaRabbitServer2Queue" Multiplicity="*" />
    <ReferentialConstraint>
      <Principal Role="MetaRabbitServer">
```

<PropertyRef Name="RabbitServerId" />
</Principal>
<Dependent Role="MetaRabbitServer2Queue">
<PropertyRef Name="RabbitServerId" />
</Dependent>
</ReferentialConstraint>
</Association>
<EntityContainer Name="MetaAzureModelStoreContainer">
<EntitySet Name="MetaRabbitQueue" EntityType="Self.MetaRabbitQueue" Schema="dbo" store:Type="Tables" />
<EntitySet Name="MetaRabbitServer" EntityType="Self.MetaRabbitServer" Schema="dbo" store:Type="Tables" />
<EntitySet Name="MetaRabbitServer2Queue" EntityType="Self.MetaRabbitServer2Queue" Schema="dbo"
store:Type="Tables" />
<AssociationSet Name="FK_MetaRabbitServer2Queue_MetaRabbitQueue"
Association="Self.FK_MetaRabbitServer2Queue_MetaRabbitQueue">
<End Role="MetaRabbitQueue" EntitySet="MetaRabbitQueue" />
<End Role="MetaRabbitServer2Queue" EntitySet="MetaRabbitServer2Queue" />
</AssociationSet>
<AssociationSet Name="FK_MetaRabbitServer2Queue_MetaRabbitServer"
Association="Self.FK_MetaRabbitServer2Queue_MetaRabbitServer">
<End Role="MetaRabbitServer" EntitySet="MetaRabbitServer" />
<End Role="MetaRabbitServer2Queue" EntitySet="MetaRabbitServer2Queue" />
</AssociationSet>
</EntityContainer>
</Schema>
</edmx:StorageModels>
<!-- CSDL content -->

<edmx:ConceptualModels>
<Schema Namespace="MetaAzureModel" Alias="Self" annotation:UseStrongSpatialTypes="false"
xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
xmlns:customannotation="http://schemas.microsoft.com/ado/2013/11/edm/customannotation"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm">
<EntityType Name="MetaRabbitQueue">
<Key>
<PropertyRef Name="RabbitQueueId" />
</Key>
<Property Name="RabbitQueueId" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
<Property Name="RabbitQueueName" Type="String" MaxLength="100" FixedLength="false" Unicode="false" />
<NavigationProperty Name="MetaRabbitServer2Queue" Relationship="Self.FK_MetaRabbitServer2Queue_MetaRabbitQueue"
FromRole="MetaRabbitQueue" ToRole="MetaRabbitServer2Queue" />
</EntityType>
<EntityType Name="MetaRabbitServer">
<Key>
<PropertyRef Name="RabbitServerId" />
</Key>
<Property Name="RabbitServerId" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
<Property Name="Environment" Type="String" MaxLength="50" FixedLength="false" Unicode="false" Nullable="false" />
<Property Name="FriendlyName" Type="String" MaxLength="50" FixedLength="false" Unicode="false" Nullable="false" />
<Property Name="WebHostName" Type="String" MaxLength="200" FixedLength="false" Unicode="false" Nullable="false" />
<Property Name="Username" Type="String" MaxLength="50" FixedLength="false" Unicode="false" Nullable="false" />
<Property Name="ServerHostName" Type="String" MaxLength="200" FixedLength="false" Unicode="false" />
<Property Name="ServerHostPortNumber" Type="Int32" Nullable="false" />
<Property Name="SortOrder" Type="Int32" Nullable="false" />

<NavigationProperty Name="MetaRabbitServer2Queue" Relationship="Self.FK_MetaRabbitServer2Queue_MetaRabbitServer" FromRole="MetaRabbitServer" ToRole="MetaRabbitServer2Queue" />
</EntityType>
<EntityType Name="MetaRabbitServer2Queue">
<Key>
<PropertyRef Name="RabbitId" />
</Key>
<Property Name="RabbitId" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity" />
<Property Name="RabbitQueueId" Type="Int32" Nullable="false" />
<Property Name="RabbitServerId" Type="Int32" Nullable="false" />
<Property Name="MaxThreshold" Type="Int32" Nullable="false" />
<Property Name="IsAlert" Type="Boolean" Nullable="false" />
<NavigationProperty Name="MetaRabbitQueue" Relationship="Self.FK_MetaRabbitServer2Queue_MetaRabbitQueue" FromRole="MetaRabbitServer2Queue" ToRole="MetaRabbitQueue" />
<NavigationProperty Name="MetaRabbitServer" Relationship="Self.FK_MetaRabbitServer2Queue_MetaRabbitServer" FromRole="MetaRabbitServer2Queue" ToRole="MetaRabbitServer" />
</EntityType>
<Association Name="FK_MetaRabbitServer2Queue_MetaRabbitQueue">
<End Role="MetaRabbitQueue" Type="Self.MetaRabbitQueue" Multiplicity="1" />
<End Role="MetaRabbitServer2Queue" Type="Self.MetaRabbitServer2Queue" Multiplicity="*" />
<ReferentialConstraint>
<Principal Role="MetaRabbitQueue">
<PropertyRef Name="RabbitQueueId" />
</Principal>
<Dependent Role="MetaRabbitServer2Queue">
<PropertyRef Name="RabbitQueueId" />

```
        </Dependent>
      </ReferentialConstraint>
    </Association>
    <Association Name="FK_MetaRabbitServer2Queue_MetaRabbitServer">
      <End Role="MetaRabbitServer" Type="Self.MetaRabbitServer" Multiplicity="1" />
      <End Role="MetaRabbitServer2Queue" Type="Self.MetaRabbitServer2Queue" Multiplicity="*" />
      <ReferentialConstraint>
        <Principal Role="MetaRabbitServer">
          <PropertyRef Name="RabbitServerId" />
        </Principal>
        <Dependent Role="MetaRabbitServer2Queue">
          <PropertyRef Name="RabbitServerId" />
        </Dependent>
      </ReferentialConstraint>
    </Association>
    <EntityContainer Name="MetaAzureEntities" annotation:LazyLoadingEnabled="true">
      <EntitySet Name="MetaRabbitQueues" EntityType="Self.MetaRabbitQueue" />
      <EntitySet Name="MetaRabbitServers" EntityType="Self.MetaRabbitServer" />
      <EntitySet Name="MetaRabbitServer2Queue" EntityType="Self.MetaRabbitServer2Queue" />
      <AssociationSet Name="FK_MetaRabbitServer2Queue_MetaRabbitQueue"
Association="Self.FK_MetaRabbitServer2Queue_MetaRabbitQueue">
        <End Role="MetaRabbitQueue" EntitySet="MetaRabbitQueues" />
        <End Role="MetaRabbitServer2Queue" EntitySet="MetaRabbitServer2Queue" />
      </AssociationSet>
      <AssociationSet Name="FK_MetaRabbitServer2Queue_MetaRabbitServer"
Association="Self.FK_MetaRabbitServer2Queue_MetaRabbitServer">
```

```
        <End Role="MetaRabbitServer" EntitySet="MetaRabbitServers" />
        <End Role="MetaRabbitServer2Queue" EntitySet="MetaRabbitServer2Queue" />
      </AssociationSet>
    </EntityContainer>
  </Schema>
</edmx:ConceptualModels>
<!-- C-S mapping content -->
<edmx:Mappings>
  <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
    <EntityContainerMapping StorageEntityContainer="MetaAzureModelStoreContainer" CdmEntityContainer="MetaAzureEntities">
      <EntitySetMapping Name="MetaRabbitQueues">
        <EntityTypeMapping TypeName="MetaAzureModel.MetaRabbitQueue">
          <MappingFragment StoreEntitySet="MetaRabbitQueue">
            <ScalarProperty Name="RabbitQueueId" ColumnName="RabbitQueueId" />
            <ScalarProperty Name="RabbitQueueName" ColumnName="RabbitQueueName" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
      <EntitySetMapping Name="MetaRabbitServers">
        <EntityTypeMapping TypeName="MetaAzureModel.MetaRabbitServer">
          <MappingFragment StoreEntitySet="MetaRabbitServer">
            <ScalarProperty Name="RabbitServerId" ColumnName="RabbitServerId" />
            <ScalarProperty Name="Environment" ColumnName="Environment" />
            <ScalarProperty Name="FriendlyName" ColumnName="FriendlyName" />
            <ScalarProperty Name="WebHostName" ColumnName="WebHostName" />
            <ScalarProperty Name="Username" ColumnName="Username" />
```

```
            <ScalarProperty Name="ServerHostName" ColumnName="ServerHostName" />
            <ScalarProperty Name="ServerHostPortNumber" ColumnName="ServerHostPortNumber" />
            <ScalarProperty Name="SortOrder" ColumnName="SortOrder" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
      <EntitySetMapping Name="MetaRabbitServer2Queue">
        <EntityTypeMapping TypeName="MetaAzureModel.MetaRabbitServer2Queue">
          <MappingFragment StoreEntitySet="MetaRabbitServer2Queue">
            <ScalarProperty Name="RabbitId" ColumnName="RabbitId" />
            <ScalarProperty Name="RabbitQueueId" ColumnName="RabbitQueueId" />
            <ScalarProperty Name="RabbitServerId" ColumnName="RabbitServerId" />
            <ScalarProperty Name="MaxThreshold" ColumnName="MaxThreshold" />
            <ScalarProperty Name="IsAlert" ColumnName="IsAlert" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
    </EntityContainerMapping>
  </Mapping>
 </edmx:Mappings>
</edmx:Runtime>
<!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
<Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">
 <Connection>
  <DesignerInfoPropertySet>
   <DesignerProperty Name="MetadataArtifactProcessing" Value="EmbedInOutputAssembly" />
```

```
      </DesignerInfoPropertySet>
    </Connection>
    <Options>
      <DesignerInfoPropertySet>
        <DesignerProperty Name="ValidateOnBuild" Value="true" />
        <DesignerProperty Name="EnablePluralization" Value="true" />
        <DesignerProperty Name="IncludeForeignKeysInModel" Value="true" />
        <DesignerProperty Name="UseLegacyProvider" Value="false" />
        <DesignerProperty Name="CodeGenerationStrategy" Value="None" />
      </DesignerInfoPropertySet>
    </Options>
    <!-- Diagram content (shape and connector positions) -->
    <Diagrams></Diagrams>
  </Designer>
</edmx:Edmx>
```

MetaAzure.edmx.diagram (Monitoring.RabbitMQAlerter\MetaAzure.edmx.diagram):

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
 <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <edmx:Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">
    <!-- Diagram content (shape and connector positions) -->
    <edmx:Diagrams>
      <Diagram DiagramId="cb65bafd998343d8a45b9104206cd79b" Name="Diagram1">
        <EntityTypeShape EntityType="MetaAzureModel.MetaRabbitQueue" Width="1.5" PointX="0.75" PointY="1.375"
IsExpanded="true" />
```

```
        <EntityTypeShape EntityType="MetaAzureModel.MetaRabbitServer" Width="1.5" PointX="0.75" PointY="4.75"
IsExpanded="true" />
        <EntityTypeShape EntityType="MetaAzureModel.MetaRabbitServer2Queue" Width="1.5" PointX="3" PointY="1"
IsExpanded="true" />
        <AssociationConnector Association="MetaAzureModel.FK_MetaRabbitServer2Queue_MetaRabbitQueue"
ManuallyRouted="false" />
        <AssociationConnector Association="MetaAzureModel.FK_MetaRabbitServer2Queue_MetaRabbitServer"
ManuallyRouted="false" />
      </Diagram>
    </edmx:Diagrams>
  </edmx:Designer>
</edmx:Edmx>
```

MetaAzure.tt (Monitoring.RabbitMQAlerter\MetaAzure.tt):

```
<#@ template language="C#" debug="false" hostspecific="true"#>
<#@ include file="EF6.Utility.CS.ttinclude"#><#@
 output extension=".cs"#><#

const string inputFile = @"MetaAzure.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 fileManager = EntityFrameworkTemplateFileManager.Create(this);
var itemCollection = new EdmMetadataLoader(textTransform.Host, textTransform.Errors).CreateEdmItemCollection(inputFile);
var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);
```

```
if (!typeMapper.VerifyCaseInsensitiveTypeUniqueness(typeMapper.GetAllGlobalItems(itemCollection), inputFile))
{
    return string.Empty;
}

WriteHeader(codeStringGenerator, fileManager);

foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>(itemCollection))
{
    fileManager.StartNewFile(entity.Name + ".cs");
    BeginNamespace(code);
#>
<#=codeStringGenerator.UsingDirectives(inHeader: false)#>
<#=codeStringGenerator.EntityClassOpening(entity)#>
{
<#
    var propertiesWithDefaultValues = typeMapper.GetPropertiesWithDefaultValues(entity);
    var collectionNavigationProperties = typeMapper.GetCollectionNavigationProperties(entity);
    var complexProperties = typeMapper.GetComplexProperties(entity);

    if (propertiesWithDefaultValues.Any() || collectionNavigationProperties.Any() || complexProperties.Any())
    {
#>
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
```

```
    public <#=code.Escape(entity)#>()
    {
<#
        foreach (var edmProperty in propertiesWithDefaultValues)
        {
#>
        this.<#=code.Escape(edmProperty)#> = <#=typeMapper.CreateLiteral(edmProperty.DefaultValue)#>;
<#
        }

        foreach (var navigationProperty in collectionNavigationProperties)
        {
#>
        this.<#=code.Escape(navigationProperty)#> = new
HashSet<<#=typeMapper.GetTypeName(navigationProperty.ToEndMember.GetEntityType())#>>();
<#
        }

        foreach (var complexProperty in complexProperties)
        {
#>
        this.<#=code.Escape(complexProperty)#> = new <#=typeMapper.GetTypeName(complexProperty.TypeUsage)#>();
<#
        }
#>
    }
```

```
<#
    }

    var simpleProperties = typeMapper.GetSimpleProperties(entity);
    if (simpleProperties.Any())
    {
        foreach (var edmProperty in simpleProperties)
        {
#>
    <#=codeStringGenerator.Property(edmProperty)#>
<#
        }
    }

    if (complexProperties.Any())
    {
#>

<#
        foreach(var complexProperty in complexProperties)
        {
#>
    <#=codeStringGenerator.Property(complexProperty)#>
<#
        }
```

```
    }

    var navigationProperties = typeMapper.GetNavigationProperties(entity);
    if (navigationProperties.Any())
    {
#>

<#
        foreach (var navigationProperty in navigationProperties)
        {
            if (navigationProperty.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many)
            {
#>
    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
<#
            }
#>
    <#=codeStringGenerator.NavigationProperty(navigationProperty)#>
<#
        }
    }
#>
}
<#
    EndNamespace(code);
}
```

```
foreach (var complex in typeMapper.GetItemsToGenerate<ComplexType>(itemCollection))
{
    fileManager.StartNewFile(complex.Name + ".cs");
    BeginNamespace(code);
#>
<#=codeStringGenerator.UsingDirectives(inHeader: false, includeCollections: false)#>
<#=Accessibility.ForType(complex)#> partial class <#=code.Escape(complex)#>
{
<#
    var complexProperties = typeMapper.GetComplexProperties(complex);
    var propertiesWithDefaultValues = typeMapper.GetPropertiesWithDefaultValues(complex);

    if (propertiesWithDefaultValues.Any() || complexProperties.Any())
    {
#>
    public <#=code.Escape(complex)#>()
    {
<#
        foreach (var edmProperty in propertiesWithDefaultValues)
        {
#>
        this.<#=code.Escape(edmProperty)#> = <#=typeMapper.CreateLiteral(edmProperty.DefaultValue)#>;
<#
        }
```

```
        foreach (var complexProperty in complexProperties)
        {
#>
        this.<#=code.Escape(complexProperty)#> = new <#=typeMapper.GetTypeName(complexProperty.TypeUsage)#>();
<#
        }
#>
    }

<#
    }

    var simpleProperties = typeMapper.GetSimpleProperties(complex);
    if (simpleProperties.Any())
    {
        foreach(var edmProperty in simpleProperties)
        {
#>
    <#=codeStringGenerator.Property(edmProperty)#>
<#
        }
    }

    if (complexProperties.Any())
    {
#>
```

```
<#
        foreach(var edmProperty in complexProperties)
        {
#>
    <#=codeStringGenerator.Property(edmProperty)#>
<#
        }
    }
#>
}
<#
    EndNamespace(code);
}

foreach (var enumType in typeMapper.GetEnumItemsToGenerate(itemCollection))
{
    fileManager.StartNewFile(enumType.Name + ".cs");
    BeginNamespace(code);
#>
<#=codeStringGenerator.UsingDirectives(inHeader: false, includeCollections: false)#>
<#
    if (typeMapper.EnumIsFlags(enumType))
    {
#>
[Flags]
```

```
<#
    }
#>
<#=codeStringGenerator.EnumOpening(enumType)#>
{
<#
    var foundOne = false;

    foreach (MetadataItem member in typeMapper.GetEnumMembers(enumType))
    {
        foundOne = true;
#>
    <#=code.Escape(typeMapper.GetEnumMemberName(member))#> = <#=typeMapper.GetEnumMemberValue(member)#>,
<#
    }

    if (foundOne)
    {
        this.GenerationEnvironment.Remove(this.GenerationEnvironment.Length - 3, 1);
    }
#>
}
<#
    EndNamespace(code);
}
```

```
fileManager.Process();

#>
<#+

public void WriteHeader(CodeStringGenerator codeStringGenerator, EntityFrameworkTemplateFileManager fileManager)
{
    fileManager.StartHeader();
#>
//------------------------------------------------------------------------------
// <auto-generated>
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine1")#>
//
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine2")#>
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine3")#>
// </auto-generated>
//------------------------------------------------------------------------------
<#=codeStringGenerator.UsingDirectives(inHeader: true)#>
<#+
    fileManager.EndBlock();
}

public void BeginNamespace(CodeGenerationTools code)
{
    var codeNamespace = code.VsNamespaceSuggestion();
    if (!String.IsNullOrEmpty(codeNamespace))
```

```
    {
#>
namespace <#=code.EscapeNamespace(codeNamespace)#>
{
<#+
        PushIndent("    ");
    }
}

public void EndNamespace(CodeGenerationTools code)
{
    if (!String.IsNullOrEmpty(code.VsNamespaceSuggestion()))
    {
        PopIndent();
#>
}
<#+
    }
}

public const string TemplateId = "CSharp_DbContext_Types_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 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 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 static string FixNamespaces(string typeName)
{
    return typeName.Replace("System.Data.Spatial.", "System.Data.Entity.Spatial.");
}

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);
    }
}
#>
```

MetaRabbitQueue.cs (Monitoring.RabbitMQAlerter\MetaRabbitQueue.cs):

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Monitoring.RabbitMQAlerter
{
    using System;
    using System.Collections.Generic;

    public partial class MetaRabbitQueue
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
```

```
        public MetaRabbitQueue()
        {
            this.MetaRabbitServer2Queue = new HashSet<MetaRabbitServer2Queue>();
        }

        public int RabbitQueueId { get; set; }
        public string RabbitQueueName { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<MetaRabbitServer2Queue> MetaRabbitServer2Queue { get; set; }
    }
}
```

MetaRabbitServer.cs (Monitoring.RabbitMQAlerter\MetaRabbitServer.cs):

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Monitoring.RabbitMQAlerter
{
    using System;
```

```
    using System.Collections.Generic;


    public partial class MetaRabbitServer
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]
        public MetaRabbitServer()
        {
            this.MetaRabbitServer2Queue = new HashSet<MetaRabbitServer2Queue>();
        }


        public int RabbitServerId { get; set; }
        public string Environment { get; set; }
        public string FriendlyName { get; set; }
        public string WebHostName { get; set; }
        public string Username { get; set; }
        public string ServerHostName { get; set; }
        public int ServerHostPortNumber { get; set; }
        public int SortOrder { get; set; }


        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<MetaRabbitServer2Queue> MetaRabbitServer2Queue { get; set; }
    }
}
```

MetaRabbitServer2Queue.cs (Monitoring.RabbitMQAlerter\MetaRabbitServer2Queue.cs):

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Monitoring.RabbitMQAlerter
{
    using System;
    using System.Collections.Generic;

    public partial class MetaRabbitServer2Queue
    {
        public int RabbitId { get; set; }
        public int RabbitQueueId { get; set; }
        public int RabbitServerId { get; set; }
        public int MaxThreshold { get; set; }
        public bool IsAlert { get; set; }

        public virtual MetaRabbitQueue MetaRabbitQueue { get; set; }
        public virtual MetaRabbitServer MetaRabbitServer { get; set; }
    }
}
```

Monitoring.RabbitMQAlerter.csproj (Monitoring.RabbitMQAlerter\Monitoring.RabbitMQAlerter.csproj):

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="..\packages\EntityFramework.6.4.4\build\EntityFramework.props"
Condition="Exists('..\packages\EntityFramework.6.4.4\build\EntityFramework.props')" />
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{30ED0FFB-878C-4699-A815-D9361CC29888}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>Monitoring.RabbitMQAlerter</RootNamespace>
    <AssemblyName>Monitoring.RabbitMQAlerter</AssemblyName>
    <TargetFrameworkVersion>v4.6.2</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <Deterministic>true</Deterministic>
    <NuGetPackageImportStamp>
    </NuGetPackageImportStamp>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
```

```
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089,
processorArchitecture=MSIL">
      <HintPath>..\packages\Monitoring.Email.4.1.4\lib\net40\EntityFramework.dll</HintPath>
    </Reference>
    <Reference Include="EntityFramework.SqlServer, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089,
processorArchitecture=MSIL">
      <HintPath>..\packages\Monitoring.Email.4.1.4\lib\net40\EntityFramework.SqlServer.dll</HintPath>
    </Reference>
    <Reference Include="Microsoft.Diagnostics.Tracing.EventSource, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=cc7b13ffcd2ddd51, processorArchitecture=MSIL">
```

<HintPath>..\packages\Microsoft.Diagnostics.Tracing.EventSource.Redist.6.0.0\lib\net461\Microsoft.Diagnostics.Tracing.EventSource.dll</HintPath>
    </Reference>
    <Reference Include="Monitoring.Email, Version=4.1.4.0, Culture=neutral, processorArchitecture=MSIL">
      <HintPath>..\packages\Monitoring.Email.4.1.4\lib\net40\Monitoring.Email.dll</HintPath>
    </Reference>
    <Reference Include="Monitoring.Email.Entity, Version=4.1.2.0, Culture=neutral, processorArchitecture=MSIL">
      <HintPath>..\packages\Monitoring.Email.4.1.4\lib\net40\Monitoring.Email.Entity.dll</HintPath>
    </Reference>
    <Reference Include="Monitoring.UserExtensions, Version=2.0.0.0, Culture=neutral, processorArchitecture=MSIL">
      <HintPath>..\packages\Monitoring.UserExtensions.2.0.0\lib\net40\Monitoring.UserExtensions.dll</HintPath>
    </Reference>
    <Reference Include="Newtonsoft.Json, Version=13.0.0.0, Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed, processorArchitecture=MSIL">
      <HintPath>..\packages\Newtonsoft.Json.13.0.1\lib\net45\Newtonsoft.Json.dll</HintPath>
    </Reference>
    <Reference Include="RabbitMQ.Client, Version=5.0.0.0, Culture=neutral, PublicKeyToken=89e7d7c5feba84ce, processorArchitecture=MSIL">
      <HintPath>..\packages\RabbitMQ.Client.5.1.2\lib\net451\RabbitMQ.Client.dll</HintPath>
    </Reference>
    <Reference Include="Serilog, Version=2.0.0.0, Culture=neutral, PublicKeyToken=24c2f752a8e58a10, processorArchitecture=MSIL">
      <HintPath>..\packages\Serilog.3.1.1\lib\net462\Serilog.dll</HintPath>
    </Reference>
    <Reference Include="Serilog.Sinks.Console, Version=5.0.1.0, Culture=neutral, PublicKeyToken=24c2f752a8e58a10, processorArchitecture=MSIL">

<HintPath>..\packages\Serilog.Sinks.Console.5.0.1\lib\net462\Serilog.Sinks.Console.dll</HintPath>

</Reference>

<Reference Include="Serilog.Sinks.File, Version=5.0.0.0, Culture=neutral, PublicKeyToken=24c2f752a8e58a10,

processorArchitecture=MSIL">

<HintPath>..\packages\Serilog.Sinks.File.5.0.0\lib\net45\Serilog.Sinks.File.dll</HintPath>

</Reference>

<Reference Include="System" />

<Reference Include="System.Buffers, Version=4.0.3.0, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51,

processorArchitecture=MSIL">

<HintPath>..\packages\System.Buffers.4.5.1\lib\net461\System.Buffers.dll</HintPath>

</Reference>

<Reference Include="System.ComponentModel.DataAnnotations" />

<Reference Include="System.Configuration" />

<Reference Include="System.Core" />

<Reference Include="System.Diagnostics.DiagnosticSource, Version=7.0.0.2, Culture=neutral,

PublicKeyToken=cc7b13ffcd2ddd51, processorArchitecture=MSIL">

<HintPath>..\packages\System.Diagnostics.DiagnosticSource.7.0.2\lib\net462\System.Diagnostics.DiagnosticSource.dll</HintPath>

</Reference>

<Reference Include="System.Memory, Version=4.0.1.2, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51,

processorArchitecture=MSIL">

<HintPath>..\packages\System.Memory.4.5.5\lib\net461\System.Memory.dll</HintPath>

</Reference>

<Reference Include="System.Numerics" />

<Reference Include="System.Numerics.Vectors, Version=4.1.4.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a,

processorArchitecture=MSIL">

```
      <HintPath>..\packages\System.Numerics.Vectors.4.5.0\lib\net46\System.Numerics.Vectors.dll</HintPath>
    </Reference>
    <Reference Include="System.Runtime.CompilerServices.Unsafe, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL">

<HintPath>..\packages\System.Runtime.CompilerServices.Unsafe.6.0.0\lib\net461\System.Runtime.CompilerServices.Unsafe.dll</Hi
ntPath>
    </Reference>
    <Reference Include="System.Runtime.Serialization" />
    <Reference Include="System.Security" />
    <Reference Include="System.Threading.Channels, Version=7.0.0.0, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51,
processorArchitecture=MSIL">
      <HintPath>..\packages\System.Threading.Channels.7.0.0\lib\net462\System.Threading.Channels.dll</HintPath>
    </Reference>
    <Reference Include="System.Threading.Tasks.Extensions, Version=4.2.0.1, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51,
processorArchitecture=MSIL">
      <HintPath>..\packages\System.Threading.Tasks.Extensions.4.5.4\lib\net461\System.Threading.Tasks.Extensions.dll</HintPath>
    </Reference>
    <Reference Include="System.ValueTuple, Version=4.0.3.0, Culture=neutral, PublicKeyToken=cc7b13ffcd2ddd51,
processorArchitecture=MSIL">
      <HintPath>..\packages\System.ValueTuple.4.5.0\lib\net461\System.ValueTuple.dll</HintPath>
    </Reference>
    <Reference Include="System.Web" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
```

```
  <Reference Include="System.Data" />
  <Reference Include="System.Net.Http" />
  <Reference Include="System.Xml" />
</ItemGroup>
<ItemGroup>
  <Compile Include="MyDbConfiguration.cs" />
  <Compile Include="DbConnection.cs" />
  <Compile Include="MetaAzure.Context.cs">
    <AutoGen>True</AutoGen>
    <DesignTime>True</DesignTime>
    <DependentUpon>MetaAzure.Context.tt</DependentUpon>
  </Compile>
  <Compile Include="MetaAzure.cs">
    <AutoGen>True</AutoGen>
    <DesignTime>True</DesignTime>
    <DependentUpon>MetaAzure.tt</DependentUpon>
  </Compile>
  <Compile Include="MetaAzure.Designer.cs">
    <AutoGen>True</AutoGen>
    <DesignTime>True</DesignTime>
    <DependentUpon>MetaAzure.edmx</DependentUpon>
  </Compile>
  <Compile Include="MetaRabbitQueue.cs">
    <DependentUpon>MetaAzure.tt</DependentUpon>
  </Compile>
  <Compile Include="MetaRabbitServer.cs">
```

```
    <DependentUpon>MetaAzure.tt</DependentUpon>
  </Compile>
  <Compile Include="MetaRabbitServer2Queue.cs">
    <DependentUpon>MetaAzure.tt</DependentUpon>
  </Compile>
  <Compile Include="Program.cs" />
  <Compile Include="Properties\AssemblyInfo.cs" />
  <Compile Include="ViewModels\QueueAssignmentViewModel.cs" />
  <Compile Include="ViewModels\QueueToServiceViewModel.cs" />
  <Compile Include="RabbitMqService.cs" />
  <Compile Include="ViewModels\RabbitMqViewModel.cs" />
  <Compile Include="ViewModels\RabbitQueue.cs" />
  <Compile Include="ViewModels\RabbitQueueAssignment.cs" />
  <Compile Include="ViewModels\RabbitQueueDetails.cs" />
  <Compile Include="ViewModels\RabbitServer.cs" />
  <Compile Include="ViewModels\ServiceAssignmentViewModel.cs" />
  <Compile Include="Utils.cs" />
</ItemGroup>
<ItemGroup>
  <None Include="App.config" />
  <EntityDeploy Include="MetaAzure.edmx">
    <Generator>EntityModelCodeGenerator</Generator>
    <LastGenOutput>MetaAzure.Designer.cs</LastGenOutput>
  </EntityDeploy>
  <None Include="MetaAzure.edmx.diagram">
    <DependentUpon>MetaAzure.edmx</DependentUpon>
```

```
    </None>
    <None Include="packages.config" />
  </ItemGroup>
  <ItemGroup>
    <Content Include="MetaAzure.Context.tt">
      <Generator>TextTemplatingFileGenerator</Generator>
      <LastGenOutput>MetaAzure.Context.cs</LastGenOutput>
      <DependentUpon>MetaAzure.edmx</DependentUpon>
    </Content>
    <Content Include="MetaAzure.tt">
      <Generator>TextTemplatingFileGenerator</Generator>
      <DependentUpon>MetaAzure.edmx</DependentUpon>
      <LastGenOutput>MetaAzure.cs</LastGenOutput>
    </Content>
  </ItemGroup>
  <ItemGroup>
    <Service Include="{508349B6-6B84-4DF5-91F0-309BEEBAD82D}" />
  </ItemGroup>
  <ItemGroup />
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
  <Target Name="EnsureNuGetPackageBuildImports" BeforeTargets="PrepareForBuild">
    <PropertyGroup>
      <ErrorText>This project references NuGet package(s) that are missing on this computer. Use NuGet Package Restore to
download them.  For more information, see http://go.microsoft.com/fwlink/?LinkID=322105. The missing file is {0}.</ErrorText>
    </PropertyGroup>
    <Error Condition="!Exists('..\packages\EntityFramework.6.4.4\build\EntityFramework.props')"
```

```
Text="$([System.String]::Format('$(ErrorText)', '..\packages\EntityFramework.6.4.4\build\EntityFramework.props'))" />
    <Error Condition="!Exists('..\packages\EntityFramework.6.4.4\build\EntityFramework.targets')"
Text="$([System.String]::Format('$(ErrorText)', '..\packages\EntityFramework.6.4.4\build\EntityFramework.targets'))" />
    <Error
Condition="!Exists('..\packages\Microsoft.VisualStudio.SlowCheetah.4.0.50\build\Microsoft.VisualStudio.SlowCheetah.targets')"
Text="$([System.String]::Format('$(ErrorText)',
'..\packages\Microsoft.VisualStudio.SlowCheetah.4.0.50\build\Microsoft.VisualStudio.SlowCheetah.targets'))" />
  </Target>
  <Import Project="..\packages\EntityFramework.6.4.4\build\EntityFramework.targets"
Condition="Exists('..\packages\EntityFramework.6.4.4\build\EntityFramework.targets')" />
  <Import Project="..\packages\Microsoft.VisualStudio.SlowCheetah.4.0.50\build\Microsoft.VisualStudio.SlowCheetah.targets"
Condition="Exists('..\packages\Microsoft.VisualStudio.SlowCheetah.4.0.50\build\Microsoft.VisualStudio.SlowCheetah.targets')" />
</Project>
```

Monitoring.RabbitMQAlerter.sln (Monitoring.RabbitMQAlerter\Monitoring.RabbitMQAlerter.sln):

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio Version 17
VisualStudioVersion = 17.5.002.0
MinimumVisualStudioVersion = 10.0.40219.1
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "Monitoring.RabbitMQAlerter", "Monitoring.RabbitMQAlerter.csproj",
"{A3C02AF5-B660-4B7D-AABF-A2F4EEC607CA}"
EndProject
Global
 GlobalSection(SolutionConfigurationPlatforms) = preSolution
  Debug|Any CPU = Debug|Any CPU
```

```
  Release|Any CPU = Release|Any CPU
 EndGlobalSection
 GlobalSection(ProjectConfigurationPlatforms) = postSolution
  {A3C02AF5-B660-4B7D-AABF-A2F4EEC607CA}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
  {A3C02AF5-B660-4B7D-AABF-A2F4EEC607CA}.Debug|Any CPU.Build.0 = Debug|Any CPU
  {A3C02AF5-B660-4B7D-AABF-A2F4EEC607CA}.Release|Any CPU.ActiveCfg = Release|Any CPU
  {A3C02AF5-B660-4B7D-AABF-A2F4EEC607CA}.Release|Any CPU.Build.0 = Release|Any CPU
 EndGlobalSection
 GlobalSection(SolutionProperties) = preSolution
  HideSolutionNode = FALSE
 EndGlobalSection
 GlobalSection(ExtensibilityGlobals) = postSolution
  SolutionGuid = {74E11F72-432F-41CB-B3DC-7D7DB6EB3B93}
 EndGlobalSection
EndGlobal
```

MyDbConfiguration.cs (Monitoring.RabbitMQAlerter\MyDbConfiguration.cs):

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

namespace Monitoring.RabbitMQAlerter
{
    public class MyDbConfiguration : DbConfiguration
    {
```

```
        public static CustomSqlAzureExecutionStrategy LastStrategyInstance { get; private set; }


        public MyDbConfiguration()
        {
            SetExecutionStrategy("System.Data.SqlClient", () =>
            {
                var strategy = new CustomSqlAzureExecutionStrategy(7, TimeSpan.FromSeconds(30)); // Adjust the retry count and delay
as needed
                LastStrategyInstance = strategy;
                return strategy;
            });
        }
    }


    public class CustomSqlAzureExecutionStrategy : SqlAzureExecutionStrategy
    {
        private static ILogger _log = Log.ForContext<CustomSqlAzureExecutionStrategy>();


        public CustomSqlAzureExecutionStrategy(int maxRetryCount, TimeSpan maxDelay)
            : base(maxRetryCount, maxDelay)
        {
            // Log a message indicating the creation of a retry strategy instance, if desired.
            _log.Information($"Custom SQL Azure Execution Strategy initialized with maxRetryCount: {maxRetryCount}, maxDelay:
{maxDelay}.");
        }
```

```
        protected override bool ShouldRetryOn(Exception exception)
        {
            var shouldRetry = base.ShouldRetryOn(exception);

            if (shouldRetry)
            {
                _log.Warning($"Retrying due to a transient exception: {exception.Message}");
            }

            return shouldRetry;
        }
    }
}
```

packages.config (Monitoring.RabbitMQAlerter\packages.config):

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="EntityFramework" version="6.4.4" targetFramework="net462" />
  <package id="Microsoft.Diagnostics.Tracing.EventSource.Redist" version="6.0.0" targetFramework="net462" />
  <package id="Microsoft.VisualStudio.SlowCheetah" version="4.0.50" targetFramework="net462" developmentDependency="true" />
  <package id="Monitoring.Email" version="4.1.4" targetFramework="net462" />
  <package id="Monitoring.UserExtensions" version="2.0.0" targetFramework="net462" />
  <package id="Newtonsoft.Json" version="13.0.1" targetFramework="net462" />
  <package id="RabbitMQ.Client" version="5.1.2" targetFramework="net462" />
  <package id="Serilog" version="3.1.1" targetFramework="net462" />
  <package id="Serilog.Sinks.Console" version="5.0.1" targetFramework="net462" />
```

```
  <package id="Serilog.Sinks.File" version="5.0.0" targetFramework="net462" />
  <package id="System.Buffers" version="4.5.1" targetFramework="net462" />
  <package id="System.Diagnostics.DiagnosticSource" version="7.0.2" targetFramework="net462" />
  <package id="System.Memory" version="4.5.5" targetFramework="net462" />
  <package id="System.Numerics.Vectors" version="4.5.0" targetFramework="net462" />
  <package id="System.Runtime.CompilerServices.Unsafe" version="6.0.0" targetFramework="net462" />
  <package id="System.Threading.Channels" version="7.0.0" targetFramework="net462" />
  <package id="System.Threading.Tasks.Extensions" version="4.5.4" targetFramework="net462" />
  <package id="System.ValueTuple" version="4.5.0" targetFramework="net462" />
</packages>
```

Program.cs (Monitoring.RabbitMQAlerter\Program.cs):

```
using Monitoring.Email.Abstract;
using Monitoring.Email.Models;
using Monitoring.Email.Services;
using Monitoring.RabbitMQAlerter.ViewModels;
using Monitoring.UserExtensions;

using Serilog;

using System;
using System.Configuration;
using System.Linq;
using System.Data.Entity;
using System.Threading;
using System.Collections.Generic;
```

```
namespace Monitoring.RabbitMQAlerter
{
    internal class Program
    {
        private static readonly int EndHour = ConfigurationManager.AppSettings["EndHour"].ToInt();
        private static readonly RabbitMqService Rmqs = new RabbitMqService();
        private static readonly int StartHour = ConfigurationManager.AppSettings["StartHour"].ToInt();
        private static ILogger _log;

        private static string alertMessage;

        private static void AddToAlert(RabbitQueueAssignment queue, RabbitServer rabbitServer, int messageCount)
        {
            var msgStr =
                $"Server: {rabbitServer.WebHostName}<br/>Queue: {queue.RabbitQueue.QueueName}<br/>Messages:
{messageCount}<br/>Threshold: {queue.AlertThreshold}<br/><br/>";
            alertMessage += msgStr;
        }

        private static void AddToAlert(string message)
        {
            alertMessage += message + "<br/>";
        }

        private static void CheckServer(RabbitServer rabbitServer)
```

```
{
    var queueAssignments = Rmqs.GetRabbitQueueAssignments(rabbitServer.ServerId);

    _log.Information($"{queueAssignments.Count} queue assignments found");
    if (queueAssignments.Count == 0)
        return;

    var thresholdQueues = queueAssignments.Where(f => f.AlertThreshold > 0).ToList();
    _log.Information($"{thresholdQueues.Count} queue with thresholds");

    if (thresholdQueues.Count == 0)
        return;


    try
    {
        var queueInfo = Rmqs.GetQueueDetails(rabbitServer);

        if (queueInfo != null)
        {
            foreach (var queue in thresholdQueues)
            {
                var currentqueue = queueInfo.items.SingleOrDefault(d => d["name"] != null &&
                    d["name"].ToString() ==
                    queue.RabbitQueue.QueueName);
```

```
                if (currentqueue != null)
                {
                    var messageCount = currentqueue["messages"].ToInt();

                    if (messageCount >= queue.AlertThreshold)
                    {
                        _log.Warning(
                            $"WARNING - Queue {queue.RabbitQueue.QueueName} has {messageCount} messages, with a threshold of
{queue.AlertThreshold}");
                        AddToAlert(queue, rabbitServer, messageCount);
                    }
                    else
                    {
                        _log.Information(
                            $"OK - Queue {queue.RabbitQueue.QueueName} has {messageCount} messages, with a threshold of
{queue.AlertThreshold}");
                    }
                }
            }
        }

        catch (Exception ex)
        {
            var msg = $"Unable to get queue information for {rabbitServer.WebHostName}.<br/> Exception: {ex.Message}<br/>";
            _log.Information(msg);
```

```
        AddToAlert(msg);
    }
}

private static bool IsValidHour()
{
    var currentHour = DateTime.Now.Hour;

    return (currentHour >= StartHour && currentHour <= EndHour);
}

private static void Main(string[] args)
{
    // iterate through list of Rabbit Server
    // check if there are any queues with thresholds > 0
    // if yes, retrieve current queue from rabbit server
    // check if any of those queues are over threshold
    // notify if yes

    // Set the configuration for Entity Framework
    DbConfiguration.SetConfiguration(new MyDbConfiguration());

    // Configure Serilog to write to the console and a file
    _log = new LoggerConfiguration()
        .WriteTo.Console()
        .WriteTo.File("log.txt", fileSizeLimitBytes: 1024 * 1024).CreateLogger();
```

```
_log.Information("Starting RMQ Alerter");



// Check if the current hour is within the valid hours

if (!IsValidHour())

{

    _log.Information($"Alerting skipped. Only valid in hours {StartHour} to {EndHour}");

    return;

}



        // Execute checks and use a specific catch block for critical, operation-halting exceptions.

try

{

    PerformAllChecks();

}

catch (Exception ex)

{

    // Process critical, non-retryable failures here. Adjust alertMessage accordingly.

    alertMessage += $"Critical Failure Occurred:<br/>{ex.Message}<br/>{ex.StackTrace}<br/><br/>";

    _log.Error($"Critical failure encountered: {ex.Message}");

}



// Send alerts if there's anything noteworthy accumulated in alertMessage.

if (!string.IsNullOrWhiteSpace(alertMessage))

{

    SendAlert();

}
```

```
    // If the program is running in local mode, it waits for user input before exiting.
    if (Environment.UserInteractive && ConfigurationManager.AppSettings["islocal"].ToBool())
    {
        Console.ReadLine();
    }
}

/// <summary>
/// This method performs all checks on the RabbitMQ servers.
/// It attempts to fetch the server list and checks each server concurrently with integrated retry logic.
/// If the server list is successfully fetched, it logs the number of servers found and then proceeds to check each server.
/// If an unexpected error occurs during the check of a server, it logs the error and adds a message to the alert.
/// </summary>
private static void PerformAllChecks()
{
    int maxRetries = 3;
    int retryDelaySeconds = 5;
    List<RabbitServer> rabbitServers = null;

    for (int attempt = 0; attempt < maxRetries; attempt++)
    {
        try
        {
            rabbitServers = Rmqs.GetServerList();
            _log.Information($"{rabbitServers.Count} Rabbit Server(s) found");
```

```
        break; // Break out of the loop on success
    }
    catch (Exception ex)
    {
        _log.Information($"Attempt {attempt + 1} failed: {ex.Message}");
        if (attempt < maxRetries - 1)
        {
            Thread.Sleep(retryDelaySeconds * 1000); // Wait before retrying
        }
        else
        {
            _log.Error("Failed to retrieve Rabbit Server list after retries.");
            alertMessage += "Critical Failure Occurred: Failed to retrieve Rabbit Server list after retries.<br/>";
            return; // Early exit if we can't fetch servers after retries
        }
    }
}

if (rabbitServers == null)
{
    return; // Early exit if no servers were fetched
}

// Proceed to check each server immediately after a successful fetch
foreach (var server in rabbitServers)
{
```

```
        try
        {
            _log.Information($"Checking server: {server.FullDescription}");
            CheckServer(server); // Assuming this method handles its own exceptions accordingly.
        }
        catch (Exception ex)
        {
            _log.Error($"Unexpected error for server {server.FullDescription}: {ex.Message}");
            alertMessage += $"Unexpected error for server {server.FullDescription}: {ex.Message}<br/>";
        }
    }
}

/// <summary>
/// This method sends an alert email with the accumulated alert message.
/// It uses the EmailService to send the email.
/// </summary>
private static void SendAlert()
{
    _log.Information("Sending alert");
    var msgStr = $"The following queues have exceeded their threshold or had an error:<br/><br/>{alertMessage}";

    var message = new EmailViewModel
    {
        FromEmail = "dc_mad@plex.com",
        Subject = "Monitoring ETL Queue Warning",
```

```
            Message = msgStr,
            EmailType = Enumeration.EmailType.NoLog,
            RecipientString = ConfigurationManager.AppSettings["errorEmail"]
            //RecipientString = "obose.uwadiale@rockwellautomation.com"
        };

        var es = new EmailService();

        var smtp = new SMTPServerViewModel
        {
            Username = ConfigurationManager.AppSettings["smtp1Username"],
            SMTPPort = ConfigurationManager.AppSettings["smtp1Port"].ToInt(),
            SMTPServer = ConfigurationManager.AppSettings["smtp1Server"]
        };
        try
        {
            es.SendMail(message, smtp);
            _log.Information("Mail sent successfully.");
        }
        catch (Exception ex)
        {
            _log.Error($"Failed to send mail: {ex.Message}");
        }
      }
   }
}
```

AssemblyInfo.cs (Monitoring.RabbitMQAlerter\Properties\AssemblyInfo.cs):

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Monitoring.RabbitMQAlerter")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Plex Systems")]
[assembly: AssemblyProduct("Monitoring.RabbitMQAlerter")]
[assembly: AssemblyCopyright("Copyright  Plex Systems 2022")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("30ed0ffb-878c-4699-a815-d9361cc29888")]

// Version information for an assembly consists of the following four values:
```

```
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

RabbitMqService.cs (Monitoring.RabbitMQAlerter\RabbitMqService.cs):

```
using Microsoft.Win32;

using Monitoring.RabbitMQAlerter.ViewModels;

using Newtonsoft.Json.Linq;

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Web;
```

```
using Serilog;

using System.Data.Entity.Infrastructure;

namespace Monitoring.RabbitMQAlerter
{
    internal class RabbitMqService
    {
        private static ILogger _log = Serilog.Log.Logger;

        public RabbitMqViewModel GetQueueDetails(RabbitServer server)
        {
            var queue = new RabbitMqViewModel
            {
                ServerFriendlyName = server.FriendlyName,
                ServerHostName = server.ServerHostName
            };

            var queueList = server.WebHostName + "api/queues";

            var handler = new System.Net.Http.HttpClientHandler();

            var httpClient = new System.Net.Http.HttpClient(handler)
            {
                BaseAddress = new Uri(queueList),
                Timeout = new TimeSpan(0, 0, 30)
```

```
};

httpClient.DefaultRequestHeaders.Add("ContentType", "application/json");

var plainTextBytes = System.Text.Encoding.UTF8.GetBytes($"{server.Username}:{server.Password}");
var val = Convert.ToBase64String(plainTextBytes);
httpClient.DefaultRequestHeaders.Add("Authorization", "Basic " + val);

var response = httpClient.GetAsync(queueList).Result;

if (response.StatusCode == HttpStatusCode.OK)
{
    using (var stream = new StreamReader(response.Content.ReadAsStreamAsync().Result))
    {
        var content = stream.ReadToEnd();
        var jsonArray = JArray.Parse(content);

        queue.items = jsonArray.ToList();

        return queue;
    }
}

if (response.StatusCode == HttpStatusCode.Unauthorized)
{
```

```
            throw new Exception(
                $"Invalid credentials to the rabbit queue: {queueList}. Check the registry on the web server for password information or the username in the rabbit server setup");
        }



        return null;
    }

    /// <summary>
    /// Return all rabbit queues, with a flag to denote if it is assigned to the given rabbit server
    /// </summary>
    /// <param name="rabbitServerId"></param>
    /// <returns></returns>
    public List<RabbitQueueAssignment> GetRabbitQueueAssignments(int rabbitServerId)
    {
        using (var db = new MetaAzureDbContext())
        {
            var queues = db.MetaRabbitQueues
                .Select(f => new RabbitQueueAssignment
                {
                    RabbitQueue = new RabbitQueue
                    {
                        QueueName = f.RabbitQueueName,
                        RabbitQueueId = f.RabbitQueueId,
                    }
```

```
    }).ToList();


var queuesAssigned = db.MetaRabbitQueues.Where(f =>
    f.MetaRabbitServer2Queue.Any(g => g.RabbitServerId == rabbitServerId));


foreach (var q in queues)
{
    var assignedQueue =
        queuesAssigned.SingleOrDefault(f => f.RabbitQueueId == q.RabbitQueue.RabbitQueueId);


    if (assignedQueue != null)
    {
        var assigned =
            assignedQueue.MetaRabbitServer2Queue.SingleOrDefault(
                f => f.RabbitServerId == rabbitServerId);


        q.IsAssigned = true;


        if (assigned != null)
        {
            q.IsAlert = assigned.IsAlert;
            q.AlertThreshold = assigned.MaxThreshold;
        }
    }
}
```

```
        return queues.OrderBy(f => f.RabbitQueue.QueueName).ToList();
    }
}

/// <summary>
/// Get a list of rabbit servers
/// </summary>
/// <returns></returns>
public List<RabbitServer> GetServerList()
{
    // Assuming GetRabbitMqRegistryServers() does not need additional error handling in this context
    var registryServers = GetRabbitMqRegistryServers();

    List<RabbitServer> servers = new List<RabbitServer>();

    try
    {
        using (var db = new MetaAzureDbContext())
        {
            servers = db.MetaRabbitServers
                        .Where(f => f.WebHostName.StartsWith("http"))
                        .OrderBy(f => f.SortOrder).ThenBy(f => f.FriendlyName)
                        .AsEnumerable()
                        .Select(d => new RabbitServer
                        {
                            Environment = d.Environment,
```

```
                    FriendlyName = d.FriendlyName,
                    ServerHostName = d.ServerHostName,
                    ServerHostPortNumber = d.ServerHostPortNumber,
                    ServerId = d.RabbitServerId,
                    SortOrder = d.SortOrder,
                    Username = d.Username,
                    WebHostName = VirtualPathUtility.AppendTrailingSlash(d.WebHostName),
                }).ToList();

            foreach (var server in servers)
            {
                server.Password = registryServers.SingleOrDefault(f => f.Environment.ToUpper() ==
server.Environment.ToUpper())?.Password;
            }
        }
    }
    catch (RetryLimitExceededException ex)
    {
        // Log the details of the exception, specifically the inner exception
        _log.Error(ex, "A RetryLimitExceededException occurred in GetServerList");
        // Log the inner exception if it exists
        if (ex.InnerException != null)
        {
            _log.Error(ex.InnerException, "Inner exception in GetServerList");
        }
        // Depending on the severity of the exception and the criticality of this operation,
```

```
        // we might decide to rethrow, handle it silently, or perform a specific recovery action.
        // For now, let's rethrow to ensure the calling code can react appropriately.
        throw;
    }
    catch (Exception ex)
    {
        // Log any other types of exceptions here
        _log.Error(ex, "An unexpected exception occurred in GetServerList");
        // Rethrow or handle accordingly
        throw;
    }

    return servers;
}

public List<RabbitQueueDetails> LoadQueue(RabbitServer server)
{
    HttpWebRequest http = (HttpWebRequest)HttpWebRequest.Create(server.WebHostName);
    http.Credentials = new NetworkCredential(server.Username, server.Password);

    HttpWebResponse response = (HttpWebResponse)http.GetResponse();
    using (StreamReader sr = new StreamReader(response.GetResponseStream()))
    {
        string responseJson = sr.ReadToEnd();
    }
```

```
            return null;
        }

        /// <summary>
        /// Get a list of RabbitMQ Servers defined in the registry
        /// </summary>
        /// <returns></returns>
        /// <exception cref="ArgumentNullException"></exception>
        private static List<RabbitServer> GetRabbitMqRegistryServers()
        {
            var servers = new List<RabbitServer>();

            using (var hklm = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64))
            {
                const string keylocation = @"Software\Plex\CloudOps\Monitoring\RabbitMQ";

                using (var rabbitMqKeys = hklm.OpenSubKey(keylocation))
                {
                    if (rabbitMqKeys == null)
                        throw new ArgumentNullException(nameof(rabbitMqKeys), @"Software\Plex\CloudOps\Monitoring\RabbitMQ not
found in Registry");

                    foreach (var entry in rabbitMqKeys.GetSubKeyNames())
                    {
                        var itemKey = keylocation + @"\" + entry;
```

```
                using (var environmentKey = hklm.OpenSubKey(itemKey))
                {
                    if (environmentKey != null)
                    {
                        var server = new RabbitServer
                        {
                            Password = environmentKey.GetValue("Password").ToString(),

                            Environment = entry.ToUpper()
                        };

                        servers.Add(server);
                    }
                }
            }
        }

        return servers.OrderBy(d => d.SortOrder).ToList();
    }
  }
}
```

Utils.cs (Monitoring.RabbitMQAlerter\Utils.cs):

```
using Serilog;
using System;
```

```
using System.Collections.Generic;
using System.Data.SqlClient;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace Monitoring.ServiceRestarter
{
    public static class Utils
    {
        /// <summary>
        /// Get the connection string from the infrastructure.xml file. If the MetaSettings
        /// Environment is NOT local then connect to the PROUDUCTION azure database
        /// </summary>
        /// <param name="serviceId"></param>
        /// <param name="defaultDatabase"></param>
        /// <returns></returns>
        private static ILogger _log = Serilog.Log.Logger;
        public static string GetInfrastructureConnectionString(string serviceId, string defaultDatabase)
        {
            var filePath = FindInfrastructurePath();
            var tryCount = 1;
            XElement infrastructureXml = null;
```

```
while (tryCount <= 3)
{
    try
    {
        infrastructureXml = XElement.Load(filePath);
        break;
    }
    catch
    {
        Thread.Sleep(500); //wait 1/2 second to try again
        tryCount++;
    }
}

if (infrastructureXml == null)
    throw new ArgumentNullException(nameof(infrastructureXml),
        "Unable to load the infrastructure.xml file. I tried 3 times, and it failed each time.");

var service = infrastructureXml.Elements("services").Elements("databases")
    .Descendants().FirstOrDefault(d => d.Attribute("id")?.Value == serviceId);

if (service != null)
{
    try
    {
```

```
        var host = service.Attribute("host")?.Value ?? "";

        var password = service.Attribute("password")?.Value;
        var username = service.Attribute("username")?.Value;

        var sqlConnectionStringBuilder = new SqlConnectionStringBuilder
        {
            UserID = username,
            Password = password,
            DataSource = host,
            InitialCatalog = defaultDatabase
        };

        var connString = sqlConnectionStringBuilder.ConnectionString;

        return connString;
    }
    catch (Exception ex)
    {
        // Assuming a logging mechanism is available in the context
        _log.Error(ex, "Error building SQL connection string");
        throw;
    }
}

return "";
```

```
}

/// <summary>
/// Find the path to the infrastructure.xml using the appropriate search logic
/// </summary>
/// <returns></returns>
/// <remarks>
/// Searches in the following order: AppContext.BaseDirectory\inetpub\config\infrastructure,
/// AppContext.BaseDirectory\infrastructure.xml" AppContext.BaseDirectory\config\infrastructure.xml"
/// </remarks>
private static string FindInfrastructurePath()
{
    var currentPath = Path.Combine(Path.GetPathRoot(AppContext.BaseDirectory), @"inetpub\config\infrastructure.xml");
    if (File.Exists(currentPath))
        return currentPath;

    currentPath = Path.Combine(AppContext.BaseDirectory ?? string.Empty, "infrastructure.xml");
    if (File.Exists(currentPath))
        return currentPath;

    currentPath = Path.Combine(Path.GetPathRoot(AppContext.BaseDirectory) ?? string.Empty, @"config\infrastructure.xml");
    if (File.Exists(currentPath))
        return currentPath;

    currentPath = Path.Combine(Path.GetPathRoot(AppContext.BaseDirectory) ?? string.Empty, @"infrastructure.xml");
    if (File.Exists(currentPath))
```

```
                return currentPath;


            throw new NullReferenceException("Infrastructure.xml path not found");
        }
    }
}
```

QueueAssignmentViewModel.cs (Monitoring.RabbitMQAlerter\ViewModels\QueueAssignmentViewModel.cs):

```
namespace Monitoring.RabbitMQAlerter.ViewModels
{
    /// <summary>
    /// Model to accept changes to the queue assignemtn
    /// </summary>
    public class QueueAssignmentViewModel
    {
        /// <summary>
        /// true if the queue is assigned to the server
        /// </summary>
        public bool IsAssigned { get; set; }


        /// <summary>
        /// The queue id
        /// </summary>
        public int QueueId { get; set; }


        /// <summary>
```

```
        /// The server id
        /// </summary>
        public int RabbitServerId { get; set; }

        public int Threshold { get; set; }
    }
}
```

QueueToServiceViewModel.cs (Monitoring.RabbitMQAlerter\ViewModels\QueueToServiceViewModel.cs):

```
namespace Monitoring.RabbitMQAlerter.ViewModels
{
    public class QueueToServiceViewModel
    {
        public int EtlId { get; set; }
        public bool IsAssigned { get; set; }
        public int RabbitId { get; set; }
        public string ServiceName { get; set; }
    }
}
```

RabbitMqViewModel.cs (Monitoring.RabbitMQAlerter\ViewModels\RabbitMqViewModel.cs):

```
using Newtonsoft.Json.Linq;

using System.Collections.Generic;

namespace Monitoring.RabbitMQAlerter.ViewModels
```

```
{
    public class RabbitMqViewModel
    {
        public List<JToken> items { get; set; }

        /// <summary>
        /// A friendly server name
        /// </summary>
        public string ServerFriendlyName { get; set; }

        /// <summary>
        /// The host name of the server
        /// </summary>
        public string ServerHostName { get; set; }
    }
}
```

RabbitQueue.cs (Monitoring.RabbitMQAlerter\ViewModels\RabbitQueue.cs):

```
using System.ComponentModel.DataAnnotations;

namespace Monitoring.RabbitMQAlerter.ViewModels
{
    /// <summary>
    /// An object which represents a rabbit queue, agnostic of a server
    /// </summary>
    public class RabbitQueue
```

```
    {
        /// <summary>
        /// The queue name
        /// </summary>
        [Required]
        public string QueueName { get; set; }

        /// <summary>
        /// The queue id
        /// </summary>
        [Required]
        [Range(1, int.MaxValue)]
        public int RabbitQueueId { get; set; }
    }
}
```

RabbitQueueAssignment.cs (Monitoring.RabbitMQAlerter\ViewModels\RabbitQueueAssignment.cs):

```
namespace Monitoring.RabbitMQAlerter.ViewModels
{
    public class RabbitQueueAssignment
    {
        public int AlertThreshold { get; set; }
        public bool IsAlert { get; set; }
        public bool IsAssigned { get; set; }
        public RabbitQueue RabbitQueue { get; set; }
    }
```

```
}
```

RabbitQueueDetails.cs (Monitoring.RabbitMQAlerter\ViewModels\RabbitQueueDetails.cs):

```
using System.Runtime.Remoting.Messaging;

namespace Monitoring.RabbitMQAlerter.ViewModels
{
    /// <summary>
    ///
    /// </summary>
    public class RabbitQueueDetails
    {
        /// <summary>
        /// Backing queue status
        /// </summary>
        public BackingQueueStatus backing_queue_status { get; set; }

        /// <summary>
        /// Consumer utilisation percentage
        /// </summary>
        public decimal? consumer_utilisation { get; set; }

        /// <summary>
        /// Utilization percentage string
        /// </summary>
        public string consumer_utilisation_percent => $"{consumer_utilisation ?? 0:0%}";
```

```
/// <summary>
/// Consumer count
/// </summary>
public int? consumers { get; set; }

/// <summary>
/// Total messages
/// </summary>
public int? messages { get; set; }

/// <summary>
/// Number of messages ready
/// </summary>
public int? messages_ready { get; set; }

/// <summary>
/// Number of unacked messages
/// </summary>
public int? messages_unacknowledged { get; set; }

/// <summary>
/// Queue Name
/// </summary>
public string name { get; set; }
```

```
/// <summary>

/// State of the queue

/// </summary>

public string state { get; set; }



/// <summary>

/// object to represent a backing queue

/// </summary>

public class BackingQueueStatus

{

    /// <summary>

    /// average acknowledged egress rate

    /// </summary>

    public decimal avg_ack_egress_rate { get; set; }



    /// <summary>

    /// average acknowledged ingress rate

    /// </summary>

    public decimal avg_ack_ingress_rate { get; set; }



    /// <summary>

    /// Average egress rate

    /// </summary>

    public decimal avg_egress_rate { get; set; }



    /// <summary>
```

```
            /// Average ingress rate
            /// </summary>
            public decimal avg_ingress_rate { get; set; }
        }
    }
}
```

RabbitServer.cs (Monitoring.RabbitMQAlerter\ViewModels\RabbitServer.cs):

```
using System.ComponentModel.DataAnnotations;

namespace Monitoring.RabbitMQAlerter.ViewModels
{
    /// <summary>
    /// A model which represents a rabbit server
    /// </summary>
    public class RabbitServer
    {
        /// <summary>
        /// Friendly environment name
        /// </summary>
        public string Environment { get; set; }

        /// <summary>
        /// The friendly name of the server
        /// </summary>
        [Required]
```

```
public string FriendlyName { get; set; }

/// <summary>
/// Description and web host name
/// </summary>
public string FullDescription => FriendlyName + $" - ({WebHostName})";

/// <summary>
/// The password
/// </summary>
public string Password { get; set; }

/// <summary>
/// The server host
/// </summary>
[Required]
public string ServerHostName { get; set; }

/// <summary>
/// The web port number
/// </summary>
[Required]
public int ServerHostPortNumber { get; set; }

/// <summary>
/// Internal id
```

```
    /// </summary>
    public int ServerId { get; set; }

    /// <summary>
    /// Display order
    /// </summary>
    public int SortOrder { get; set; }

    /// <summary>
    /// The login username
    /// </summary>
    [Required]
    public string Username { get; set; }

    /// <summary>
    /// The virtual host
    /// </summary>
    public string VirtualHost => "monitoring";

    /// <summary>
    /// The web host name
    /// </summary>
    ///
    [Required]
    public string WebHostName { get; set; }
}
```

```
}
```

ServiceAssignmentViewModel.cs (Monitoring.RabbitMQAlerter\ViewModels\ServiceAssignmentViewModel.cs):

```
namespace Monitoring.RabbitMQAlerter.ViewModels
{
    /// <summary>
    /// Model to accept changes to the queue assignemtn
    /// </summary>
    public class ServiceAssignmentViewModel
    {
        /// <summary>
        /// true if the queue is assigned to the server
        /// </summary>
        public bool IsAssigned { get; set; }

        /// <summary>
        /// The server id
        /// </summary>
        public int ServerId { get; set; }

        /// <summary>
        /// The queue id
        /// </summary>
        public int ServiceId { get; set; }
    }
}
```