TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Pavel Volkov 180464IADB

IQUERYABLE INTERFACE IMPLEMENTATION FOR WINDOWS MANAGEMENT INSTRUMENTATION USAGE

Bachelor’s thesis

Supervisor: Jaanus Pöial
PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Pavel Volkov 180464IADB

IQUERYABLE LIIDESE REALISATSIOON WINDOWS MANAGEMENT INSTRUMENTATION KASUTAMISEKS

Bakalaureusetöö

Juhendaja: Jaanus Pöial
PhD

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Pavel Volkov

29.04.2020

Abstract

The aim of the current thesis is to develop a framework by implementing *.NET LINQ* [1] (Language integrated query) IQueryable [2] interface for Windows Management Instrumentation that helps developers make WQL (Windows Query Language) queries by using of LINQ fluent API [3] (Application Programming Interface) and get mapped *.NET* object with all the properties and methods WMI (Windows Management Instrumentation) class provides.

Development includes creation of framework that allows making WQL queries both ways locally and remotely. Framework will help to query, create, update and delete WMI objects and use methods they provide.

Development process outcome is working framework prototype available for developers in GitHub and Nuget.org [4] package repository.

This thesis is written in English and is 54 pages long, including 6 chapters, 13 figures and 18 tables.

Annotatsioon

IQUERYABLE LIIDESE REALISATSIOON WINDOWS MANAGEMENT INSTRUMENTATION KASUTAMISEKS

Käesoleva bakalaureusetöö eesmärk on Windows Management Instrumentations´i jaoks tarkvara raamistiku arendamine, mis realiseeritakse *.NET LINQ* IQueryable liidese kaudu.

Arendusprotsessi käigus luuakse raamistik, mis võimaldab teha WQL päringuid nii lokaalselt kui ka võrgu kaudu. Raamistik aitab pärida, luua, muuta ning kustutada WMI objekte ning kasutada kõik meetodeid mis antud objekt pakkub.

Arendusprotsessi tulemus on töötava raamistiku prototüüp mis on kättesaadav GitHub'i ning Nuget.org [4] paketihoidla kaudu.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 54 leheküljel, 6 peatükki, 13 joonist, 18 tabelit.

List of abbreviations and terms

API	Application Programming Interface
CRUD	Create Read Update Delete
DBMS	Database Management System
DCOM	Distributed Component Object Model
DMTF	Distributed Management Task Force
IDE	Integrated Development Environment
LINQ	Language Integrated Query
SQL	Structured Query Language
WinRM	Widows Remote Management
WMI	Windows Management Instrumentation
WQL	Windows Query Language
XML	eXtensible Markup Language

Table of contents

List of figures

List of tables

1 Introduction

1.1 Background

This thesis includes creation of Framework that helps developers making WMI [5] queries, all CRUD (Create Read Update Delete) operations with WMI objects and using all methods they provide.

The problem is that there are no good tools in both *.NET Framework* and *.NET Core* [6] that allows to make WQL [7] queries by using convenient interface, no common tools to use any way or protocol to connect to WMI and get all needed objects or information. There are many ways to make these queries manually but all of them are different and require development additional functions, composing of queries manually with all the needed checks.

During the last several years I was searching for similar tools for my projects but did not find anything that could cover all my needs. Instead of that, I found a lot of questions on forums from people who were looking for the same thing.

To eliminate the above problem as main topic of this thesis new framework prototype will be developed that will help developers making all needed queries by using *.NET LINQ* [1] fluent API [8] [3], that will replace big code blocks by just 1-2 lines of code.

1.2 Goal

Take *.NET LINQ* [1] IQueryable [2] interface and implement it for Windows Management Instrumentation [5], create a context for WMI namespace and use Visitor pattern [9] to translate LINQ Expressions to WQL queries and map response to appropriate .NET class properties and delegate methods.

This library will allow:

- Usage of WMI [5] as a context of classes available
- Filter objects using .NET LINQ fluent API [8] [3]
- Using different protocols to connect to WMI namespaces using same approach

Also, create a context implementation for Windows default namespace – “root\CIMV2”.

1.3 Methodology

Analysis of the current situation, choosing the core needs of developers to be satisfied, implementing corresponding developer tools and measuring the effect of usage of these tools.

Goal reaching will be done via creation of several libraries:

1. Library that will have context class as a main interface to work with and all required attribute classes to configure classes and their properties.
2. WQL [7] query translator library that will be used by all the drivers.
3. Driver library that is adding support to connect to WMI namespace by using WinRM [10] (Widows Remote Management).
4. Driver library that is adding support to connect to WMI namespace by using DCOM [11] (Distributed Component Object Model), this driver supports extended WQL.
5. Context containing library, that represents default windows namespace “root\CIMV2” with all its classes.

The functionality is divided to several libraries to make this framework more flexible and is providing possibility to extend it by adding new classes, contexts and connection drivers without modification of main library. Also, this approach supports Open-Closed Principle [12].

2 Problem description

2.1 Existing solutions

It is a quite common when application due to its needs require to gather some information about computer hardware or from some software that uses WMI [5] namespaces and classes as an API [13], or application could even control some things using same technology.

This can be achieved using different ways and protocols. For example, it can be done via WinRM [10] or DCOM [11] protocols, it is possible to use existing libraries in framework or some command line utilities. All these approaches are suitable. They are different and requires a lot of manual work that could be automated and implemented more standard way.

Currently existing solution in .NET is to use one of 2 available protocols and create all required objects manually. First of all connection needs to be created that providing all the configuration and then it is needed to compose WQL [7] queries manually, it means that self-developed class libraries needs to be created each time to meet business logic needs. This approach is not the best one because each implementation is different, and everything needs to be done manually and if there is a need to switch connection protocol completely new library needs to be developed because approaches used with different protocols are different.

There are other solutions in GitHub [14] that provides helper classes for working with WMI but they are limited with one connection method only and they are limiting functionality of WMI [5].

2.1.1 ORMi

This is the example of WMI helper that helps to map query results to .NET class and use all the CRUD operation on received object. Also, there is possibility to add methods to own classes and use them with WMI objects.

ORMi is using DCOM protocol only and does not have possibility to extend its functionality by adding different connection drivers.

This solution still requires composing WQL queries manually in case of filtering need and does not support extended WQL [15]. Also, each method invocation creates separate connection that should to be avoided.

2.2 Scope of WMI Queryable framework solution

The scope is to develop WMI Queryable framework prototype by using .NET LINQ [1]. This solution will provide single context object where it is possible to have any of the WMI object that can be enumerated or filtered by using fluent API [3] and get the result in required format.

Object mapper will map all the properties and declared delegate methods automatically so that there will not be needed to create any method body.

Framework supports dependency injection container available in ASP.NET Core [16] [17] so the connection can be easily created and added to the dependency injection container.

All the classes will be available via the collection object that implements .NET LINQ IQueryable interface, so it provides all the benefits it has. All the LINQ fluent API [8] [3] expressions will be translated to WQL queries in the background automatically and will provide all the required objects. Quite lengthy code blocks can be replaced with just one-liners.

This framework provides common approach for all the WMI related activities and eliminates almost all human errors because it is strongly typed.

Framework will make development process a lot easier when WMI interface needs to be used.

It will be supported by all languages supporting any of .NET implementations including scripting language PowerShell and its cross-platform implementation PowerShell Core.

3 Solution analysis

3.1 Researched material

3.1.1 Article series “LINQ: BUILDING AN IQUERYABLE PROVIDER SERIES”

This is very good article series [18] describes basics of implementation of LINQ IQueryable and IQueryProvider interfaces on simplified example of its implementation for SQL [19] (Structured Query Language). Unfortunately, this article series has been removed, but still accessible via the “Way Back Machine”.

These articles helped to understand the basics of LINQ IQueryable interface implementation principles and its benefits.

3.1.2 Microsoft Entity Framework

Microsoft has developed great framework called Entity Framework [20], this framework main idea is implementation of LINQ IQueryable interface for SQL [19] server backends by using different drivers for different DBMS (Database Management System) and mapping results to appropriate objects.

This approach is very standardized, has common methods to work with any of the data bases and different object models.

The Entity Framework example gave very good idea to use similar approach for working with WMI objects and methods.

3.1.3 Book “Design Patterns: Elements of Reusable Object-Oriented Software”

This is a great collection of design patters that are used widely and verified during many years of programming. This book [21] explains a lot about the patterns and the best practices of using them. It helps to implement algorithms in correct and more efficient way.

3.1.4 Online Course “Design Patterns in C# and .NET”

This online course [22], created by Dmitri Nesteruk, is amazing, it describes all the design patterns from book “Design Patterns: Elements of Reusable Object-Oriented Software”, how to use them in the best way in C# programming language.

This book helps developer to understand design patterns using a lot of example exercises.

3.1.5 Microsoft documentation for “IQueryable Interface”

This material [2] helps to check all the methods, possibilities and structure of .NET IQueryable interface that is required in current development.

3.1.6 Microsoft article “Connecting to WMI Remotely with C#”

This article [23] explains how to make queries and get date from the WMI, comparison of two approaches that are used in drivers for the framework. These approaches are low level and require doing everything manually.

3.2 Technology selection for problem solution

The goal of this thesis defines quite narrow choice of technologies as the goal is to provide convenience for developers who use .NET environment. Nowadays .NET has several .NET framework implementations that support different languages.

3.2.1 Framework selection

As the framework is designed to improve development process in .NET environment it must be developed in one of the implementations of the .NET Framework.

The choice is following:

Table 1 .NET Framework implementations comparison

.NET Implementation	Description
.NET Framework [24]	This is the first implementation of .NET Framework developed by Microsoft in year 2002 and nowadays has become obsolete as Microsoft has stopped development of this product. This is Windows-based framework that can be used only for Windows applications. The latest version was released in July 2019.
.NET Core [25]	Is the new and cross-platform implementation of the framework that .NET Foundation released first time in year 2016 and the latest stable version was released in February 2020. This framework is actively being developed and updated.
.NET Standard [26]	Special implementation that is designed for creation of Class Library projects that could be used in any implementation of .NET implementation, Xamarin, Unity, Mono, etc… This framework is actively being developed and updated.

Referring to the Table 1 the best .NET framework implementation for library development is .NET Standard.

The latest version of .NET Standard is 2.1 that is not supported by obsolete Windows-based .NET Framework implementation anymore. As WMI Queryable Framework must be usable in all implementations including obsolete .NET Framework, as it is still widely used and Windows PowerShell is working in .NET Framework environment, the WMI Query Framework will be using .NET Standard version 2.0 as it is supported by all implementations of .NET frameworks.

3.2.2 Language Selection

The selected .NET framework implementation supports several languages to be used.

Table 2 .NET Programming Languages comparison [27]

Language	**Experience**	**Description**
C#	Very Good	C# (pronounced "C sharp") is a simple, modern, object-oriented, and type-safe programming language. Its roots in the C family of languages makes C# immediately familiar to C, C++, Java, and JavaScript programmers.
Visual Basic	Good	Visual Basic is an approachable language with a simple syntax for building type-safe, object-oriented apps.
F#	Weak	F# (pronounced "F sharp") is a cross-platform, open-source, functional programming language for .NET. It also includes object-oriented and imperative programming.

As per described in Table 2 the best and more convenient language to develop WMI Queryable Framework is C#, as author has more experience with it and this is more modern and suitable language for such project.

3.2.3 Approach selection

The .NET frameworks have many options for implementation different libraries.

One of the options is helper classes that can contain set of static methods, but in this case, it will be set of independent functions that will implement functionality. This approach will not provide us complete solution.

Another approach is object based with its method. This solution is better that the one above but will not provide us such flexibility and fluent API [3] to work with sets of objects.

The best and more suitable option is to implement .NET LINQ IQueryable interface that provides us convenient set of extension methods that could be used for filtering, joining, selection of objects by using of fluent API [3].

3.3 Developer tools selection

There are a lot of IDEs (Integrated Development Environment) [28] available nowadays, here are some more popular.

Table 3 IDEs Comparison

IDE	**Description**
Visual Studio [29] + JetBrains ReSharper [30] plugin	This IDE is developed by Microsoft and fully supports all implementations of .NET frameworks. It has all needed tools for debugging and testing of the code. JetBrains ReSharper is providing convenience during the development process by adding a lot of automatic code generation features.
Visual Studio Code [29]	This is open source IDE [28] that is more lightweight comparing to Visual Studio and supports extensions. It is possible to configure Visual Studio Code for almost all needs with the use of extensions. But this IDE requires much configuration.
JetBrains Rider [31]	JetBrains Rider is cross-platform IDE for development in .NET environment. This has all the functionality that ReSharper has and all the debugging and testing required tools.

As mentioned in Table 3 there are different tools suitable for the development in .NET, but as WMI Queryable framework is mostly oriented for Windows systems Visual Studio with ReSharper plugin was selected.

3.4 Source control management solution

There are several source code control management solutions available that are suitable for the project needs.

Table 4 Source control systems

Source control system	Description
GitHub	GitHub, Inc. is a United States-based global company that provides hosting for software development version control using Git. As of January 2020, GitHub reports having over 40 million users and more than 100 million repositories (including at least 28 million public repositories) [32]. This solution is mainly used for keeping source code for the Nuget.org [4] packages.
Bitbucket	Bitbucket is a web-based version control repository hosting service owned by Atlassian, for source code and development projects that use either Mercurial or Git revision control systems [33].
GitLab	GitLab is a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and continuous integration/continuous deployment pipeline features, using an open-source license, developed by GitLab Inc [34].

As per Table 4 all described source control management solutions are suitable but as GitHub is the default source control management solution for Nuget.org [4] packages and one of the goals is to have a WMI Queryable framework available as the Nuget.org package the GitHub is the best suitable option.

3.5 Solution architecture

The WMI Queryable framework architecture is designed according to clean architecture principles, and according to SOLID [12] principles mentioned in Table 5.

Table 5 SOLID Principles [12]

Name	**Description**
Single Responsibility Principle	A class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
Open–closed principle	Software entities should be open for extension but closed for modification.
Liskov substitution principle	Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
Interface segregation principle	Many client-specific interfaces are better than one general-purpose interface.
Dependency inversion principle	One should “depend upon abstractions, [not] concretions.”

The WMI Queryable framework is divided into several separate projects:

- VNetDev.WmiQueryableCore – The project represents main WMI Queryable library that provides base WMI context object, attributes, exception classes and interfaces.
- VNetDev.WmiQueryableCore.WqlTranslator – The projects provide WMI translation functionality.
- VNetDev.WmiQueryableCore.WqlTranslator.Abstraction – This project contains a set of interfaces represent WqlTranslator object and its elements.
- VNetDev.WmiQueryableCore.Cim – The WMI Queryable driver provides communication over WinRM [10] protocol.
- VNetDev.WmiQueryableCore.DCom - The WMI Queryable driver provides communication over DOM [11] protocol.
- VNetDev.WmiQueryableCore.CIMv2 – The project that extends Base WMI context class by implementing all the classes provided in Windows default WMI namespace “root\CIMV2”.

As all the depended classes are used via interfaces WMI Queryable framework provides possibility to add new drivers and context classes very easily.

3.6 Analysis summary

The analysis has covered suitable options of .NET implementations, languages and IDEs and as the result of analysis best options have been selected to develop WMI Queryable framework in better and the most efficient way.

The technology chosen provides possibility to use this framework not only in programming languages but also in the scripts as the .NET Standard implementation supports all the .NET framework implementations.

Design allows WMI Queryable framework to be extended very easily that also implements OCP (Open-Closed principle) [12].

As the development language chosen C# development process will be easier because author has quite good experience in that language.

4 Description of solution

The WMI Queryable framework is divided into several sub-projects, each project has its own goal.

4.1 “VNetDev.WmiQueryableCore” library

This library has the main WmiContext class that must be inherited in order to create own WMI context that will represent set of classes or complete WMI Namespace.

This project, in addition to base WMI context class, contains several supportive classes listed in Table 6.

Table 6 WMI Context configuration classes

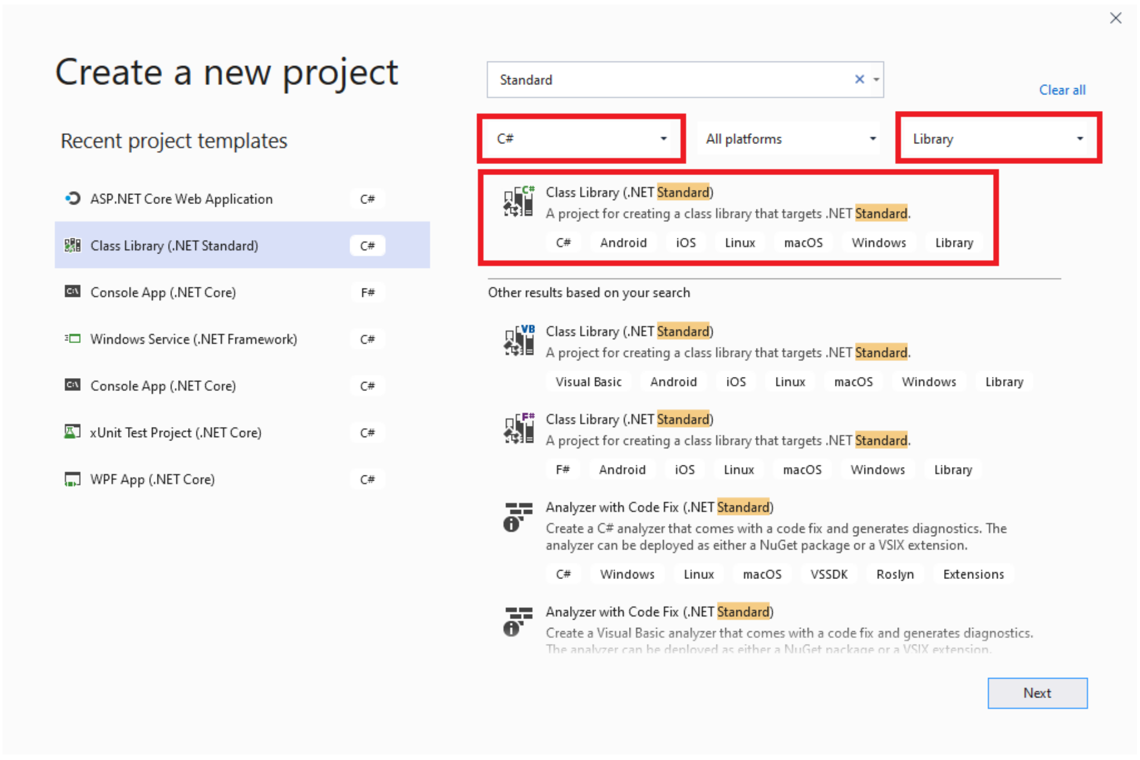
Class name	**Description**
WmiContextOptions and WmiContextOptions<TWmiContext>	Options class and its generic variant is needed for WMI Context to get connection and other initialization related options.
WmiContextOptionsBuilder and WmiContextOptionsBuilder<TWmiContext>	This class is helping to build WmiContextOpetion by using convenient fluent interface [3] and it is designed also for extension methods provided with connection drivers. It is generic implementation needed to specify type of the WMI Context class that needs to be configured. This class is implementing Builder design pattern [35].
WmiClassSet<TWmiClass>	The class that implements IQueryable [2] and IListSource [36] interfaces and represents collection of WMI Class object instances. This class needs to be used to add properties to WMI Context objects with class that represents WMI class as generic argument. As this class implements IQueryable [2] interface it is automatically providing possibility to use all the LINQ extension methods and providing and fluent interface [3].

In order to make initialization of WMI Context class instance more convenient, especially in ASP.NET Core [16] projects, this project has a class with extension method for Microsoft Dependency Injection service collection interface “IServiceCollection”.

Also, this project has set of Attribute classes to annotate things, exception classes and interface “IWmiConnection” that must be implemented in case of connection-specific driver development.

4.1.1 Project creation

As during analysis phase it was decided to use .NET Standard implementation of .NET framework and Visual Studio IDE [29] [28], it is needed to create “.NET Standard Class Library project” for development in C# language as shown on Figure 1.



Create a new project

Recent project templates

ASP.NET Core Web Application	C#
Class Library (.NET Standard)	C#
Console App (.NET Core)	F#
Windows Service (.NET Framework)	C#
Console App (.NET Core)	C#
xUnit Test Project (.NET Core)	C#
WPF App (.NET Core)	C#

Standard

Clear all

C# | All platforms | Library

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

Other results based on your search

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
Visual Basic Android iOS Linux macOS Windows Library

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
F# Android iOS Linux macOS Windows Library

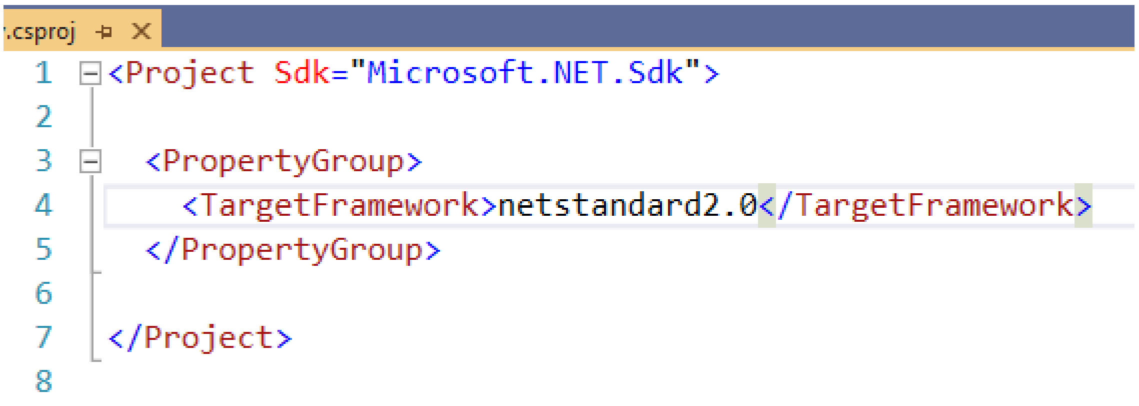
Analyzer with Code Fix (.NET Standard)
Create a C# analyzer that comes with a code fix and generates diagnostics. The analyzer can be deployed as either a NuGet package or a VSIX extension.
C# Windows Linux macOS VSSDK Roslyn Extensions

Analyzer with Code Fix (.NET Standard)
Create a Visual Basic analyzer that comes with a code fix and generates diagnostics.
The analyzer can be deployed as either a NuGet package or a VSIX extension.

Next

Figure 1 .NET Standard Class Library project creation in Visual Studio

After successful project creation it is needed to check correct version of .NET Standard framework implementation in project’s csproj [37] XML-formatted [38] (eXtensible Markup Language) file as shown on Figure 2, the “TargetFramework” tag must be set to “netstandard2.0”.



```xml
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>

</Project>
```

Figure 2 .NET Framework version configuration in project

4.1.2 WMI Context class

The main goal of this class is to provide WMI Class sets as collections with connection to the WMI [5]. This is the main interface that provides access to all added WMI Class repositories (WmiClassSets). This class implements Unit of Work design pattern [39] and WmiClassSet<TWmiCLass> implements Repository accordingly.

There are two ways to instantiate WMI Context class. First possible way is to instantiate it with providing of WmiContextOptions class instance using appropriate constructor that takes one argument of WmiContextOptions or its derived type, constructor show on Figure 3. This constructor will save the options into private property, will provide itself as a connection context. Once this is done private CreateClassSets method will be called to instantiate all the ClassSets as described in section 4.1.2.1.

```csharp
public WmiContext(WmiContextOptions options)
{
    _options = options ??
        throw new InvalidOperationException(
            "WmiContext options cannot be null!");
    _options.Connection?.SetContext(this);
    CreateClassSets();
}
```

Figure 3 WmiContext object constructor with 1 argument.

The second possible way is to instantiate WmiContext using parameterless constructor that does nothing. In this case it is possible to call Configure method later by providing configuration action for TWmiContextOptionsBuilder generic argument that represents WmiContextOptionsBuilder or its derived type. Configure method could be called only on non-configured non-disposed instance of WmiContext or its derived type object. This method will create WmiContextOptionsBuilder or its derived type object instance, type should be mentioned as generic parameter for Configure method. Once WmiContextOptionsBuilder instance created, provided action will be called to configure its options. Then built WmiContextOptions object will be assigned to WmiContext as shown on Figure 4. Once WmiContextOptions is configured private CreateClassSets method will be called to instantiate all the ClassSets as described in section 4.1.2.1.

```
public void Configure<TWmiContextOptionsBuilder>(
    Action<TWmiContextOptionsBuilder> optionsAction)
    where TWmiContextOptionsBuilder : WmiContextOptionsBuilder, new()
{
    if (_disposed || IsConfigured)
    {
        throw new InvalidOperationException(
            "This context cannot be configured anymore.");
    }

    var builder = new TWmiContextOptionsBuilder();
    if (_options != null)
    {
        builder.Options = _options;
    }

    optionsAction?.Invoke(builder);
    _options = builder.Options;
    _options.Connection?.SetContext(this);
    CreateClassSets();
}
```

Figure 4 WmiContext Configure method

4.1.2.1 CreateClassSets method.

This method goal is to instantiate all the properties of WmiClassSet<> type available in current WmiContext or its derived type instance. This method is using reflection method GetProperties() on type of current WmiContext object instance to get all available public properties, then filtering them by generic type of WmiClassSet<>, creating appropriate object instances by using Activator.CreateInstance method and assigning values to properties it requires. CreateClassSets method in shown on Figure 5.

```
private void CreateClassSets()
{
    foreach (var propertyInfo in GetType()
        .GetProperties()
        .Where(t => t.PropertyType.IsGenericType &&
                  t.PropertyType.GetGenericTypeDefinition() ==
      typeof(WmiClassSet<>)))
    {
        propertyInfo.SetValue(
            this,
            Activator.CreateInstance(
                propertyInfo.PropertyType,
                this));
    }
}
```

Figure 5 CreateClassSets method

4.1.3 WmiContext object instantiation using Dependency Injection.

One more possible way to get WmiContext or its derived type class to be instantiated is to use special IServiceCollection extension method AddWmiContext provided in static class WmiQueryableServiceCollectionExtension. This method is registering WmiClass to dependency injection container. This is the generic method that requires WmiContext type to be specified. The AddWmiContext method takes 0 to 3 arguments described in Table 7.

Table 7 AddWmiContext extension method arguments

Argument	Type	Description
optionsAction	Action<WmiContextOptionsBuilder>	This optional argument is action method that helps WmiContextOptionsBuilder to configure WmiContextOptions that will be used by instance of WmiContext or its derived type.
contextLifetime	Enum ServiceLifetime	This optional argument specifies WmiContext object instance lifetime to dependency injection container.
optionsLifetime	Enum ServiceLifetime	This optional argument specifies WmiContextOptions object instance lifetime to dependency injection container.

The benefit of it is usage inversion of control design pattern implemented in Dependency Injection container that is used widely, for example in ASP.NET Core [16]. This method allows specifying of WmiContext lifetime so that its instances will be created and kept using certain rules. Usage of this method is shown on Figure 6.

```
services.AddWmiContext<SmsWmiContext>(o =>
    o.UseCim());
```

Figure 6 Usage of AddWmiContext extension method for IServiceCollection

First, AddWmiContext extension method is using WmiContextOptionBuilder to create and configure WmiContextOptions instance and adding this configuration to Dependency Injection container with lifetime configuration provided by optionsLifetime parameter as it is required for WmiContext to be configured. Then WmiContainer or its derived type is added to Dependency injection container with lifetime configuration provided by contextLifetime parameter. The way how WmiContext class being registered in Dependency Injection container is represented on Figure 7

```
    public static IServiceCollection AddWmiContext<TContext>(
        this IServiceCollection serviceCollection,
        Action<WmiContextOptionsBuilder> optionsAction = null,
        ServiceLifetime contextLifetime = ServiceLifetime.Scoped,
        ServiceLifetime optionsLifetime = ServiceLifetime.Scoped)
        where TContext : WmiContext
    {
        serviceCollection.TryAdd(new ServiceDescriptor(
            typeof(WmiContextOptions<TContext>),
            p =>
            {
                var builder = new WmiContextOptionsBuilder<TContext>(
                    new WmiContextOptions<TContext>());
                optionsAction?.Invoke(builder);
                return builder.Options;
            },
            optionsLifetime));
        serviceCollection.Add(new ServiceDescriptor(
            typeof(WmiContextOptions),
            p => p.GetRequiredService<WmiContextOptions<TContext>>(),
            optionsLifetime));

        serviceCollection.TryAdd(new ServiceDescriptor(
            typeof(TContext), typeof(TContext), contextLifetime));
        return serviceCollection;
    }
```

Figure 7 AddWmiContext extension method

4.1.4 WmiClassSet collection class

The WmiClassSet class is one of the core elements of the systems as this class implements IQueryable<> interface and provides all the convenient functionality of it. Instance of this class requires reference to the WmiContext class instance to have a connection that is required for data querying and modification. The main querying behaviour WmiClassSet has is collecting expression tree elements provided by .NET LINQ extension methods and acting as Enumerator to get all WMI object instances constructed from WMI repository. The example of its usage shown on Figure 8.

```
foreach (var volume in context.Win32Volume
        .Where(v => v.DriveLetter != null))
    Console.WriteLine(
             $"The drive {volume.DriveLetter} has capacity of
            {volume.Capacity} bytes.");
```

Figure 8 WmiClassSet usage example

This example shows how to request data about all the available volumes in system that have DriveLetter assigned and print them out with capacity they have.

To get WMI objects constructed IQueryable requires Provider property of type implements IQueryProvider interface that provides Execute method. Execute method takes .NET LINQ Expression tree and executes it using connection driver we provide in WmiContextOptions class instance.

4.1.5 IWmiConnection interface

For WMI Queryable framework operations connectivity driver is required. This driver must to implement a set of methods declared in IWmiConnection interface. Methods are described in Table 8.

Table 8 IWmiConnection interface members

Method	**Description**
void Close()	Method that closing connection.
bool TestConnection()	Synchronous method that checks connection existence and activity. If connection exists and active it returns true, otherwise false.
Task<bool> TestConnectionAsync()	Asynchronous method that checks connection existence and activity. If connection exists and active it returns true, otherwise false.
void Delete(object wmiCLass)	Synchronous method that takes WMI class object instance and deletes it in backend.
Task DeleteAsync(object wmiCLass)	Asynchronous method that takes WMI class object instance and deletes it in backend.
void SetContext(WmiContext context)	This method takes WmiContext object instance to work with.
bool HasContext()	Method that returns true if WMI context object instance is registered and false if not.

Also, this interface inherits .NET LINQ [1] IQueryProvider interface.

4.1.6 WMI Queryable specific attributes.

WMI Queryable Framework has several attributes that are needed to configure WMI classes to be used in WmiContext. These attributes are required to configure WMI class elements. Attributes are described in Table 9.

Table 9 Attributes for WMI classes and its elements

Name	Description
WmiClassAttribute	This attribute is required for explicit specification of WMI class name in WMI namespace. This attribute is very useful as naming standards in different environments are not the same. For example, C# language uses Title case for class naming when WMI environment uses underscores for prefix separation. In order to keep naming of elements according to standards of any language this attribute is very useful.
WmiIgnorePropertyAttribute	This attribute is needed when some of the attributes need to be ignored during mapping process.
WmiMethodAttribute	This attribute can be used to set a real method name when C# class method name is different from appropriate method of class in WMI environment.
WmiPropertyAttribute	This attribute can be used to set a real property name when C# class property name is different from appropriate property of class in WMI environment.

4.2 “VNetDev.WmiQueryableCore.WqlTranslator” library

This library provides translation functionality for connection driver of WmiContext.

Project creation can be done same way as described in section 4.1.1.

The entry point is static class WqlFactory that has one public and two internal methods providing object instances. These three methods implement factory design pattern.

First and the only public method is TranslateQuery. This method takes one argument of type Expression which represents ExpressionTree to be translated and returns an object of interface type IWqlQuery described in section 4.3. This method is just a way to create an instance of WqlTranslator internal class object and to call Translate method that implements fluent interface [3] and as the result returns object of type IWqlQuery. The method TranslateQuery is shown on Figure 9.

```
public IWqlQuery TranslateQuery(Expression expression) =>
    new WqlTranslator()
        .Translate(expression);
```

Figure 9 TranslateQuery method

Two other factory methods are internal use only and they are providing instances of WqlPredicate and WqlValue described below.

4.2.1 WqlTranslator class

The WqlTranslator class has the main functionality of this library. This class extends .NET LINQ abstract class ExpressionVisitor [40] and overrides all the needed methods for expression tree translation.

This class has several private fields described in Table 10.

Table 10 Translator private fields

Name	**Type**	**Description**
_query	WqlQuery	This is the WMI Query representation object that will be returned as the translation result.
_objectStack	Stack<IWqlObject>	This is a stack of WQL [7] elements used for recursive translation of expression tree elements.
_lambdaParameterLinks	Dictionary<string, Func<string, (string, string)>>	This dictionary is used when Alias map needs to be recreated.

The alias provider region [41] of this class has one private field of type ushort and private method that provides next alias in format of "a#" as shown on Figure 10.

```
#region Alias provider

private ushort _nextAliasIndex;
private string GetNextAlias() => $"a{_nextAliasIndex++}";

#endregion
```

Figure 10 Alias provider region

Aliases and alias map are required only when join operations are used in extended queries [7] and allows to aggregate several classes into one query. Aliases are used to have unique alias for each class, as in some queries same class could be used several times. Map is needed to get required objects from the result, as the result of query with join operation is object of key-value pairs where key is assigned alias.

4.2.2 WqlQuery object

This object needs to construct WQL [7] query, keep all the joins, alias map and select functions.

All the WqlQuery elements are described in Table 11.

Table 11 WQL query elements

Object type name	**Interfaces implements**	**Description**
WqlQuery	IWqlQuery IWqlClassObject IWqlObject	This is the WMI Query representational object that will be returned as the translation result.
WqlValue	IWqlObject	This is value wrap element
WqlJoin	IWqlClassObject IWqlObject	This element represents WQL join statement to add one more class to query on condition.
WqlPredicate	IWqlPredicate IWqlObject	This object represents filtering conditions for joins and query itself.
WqlSelectDelegate	IWqlObject	This element represents output modification requested by using .NET LINQ extension method Select [42].

The main entry method is Translate, this method takes expression tree as an argument, simplifying it using available in helper Evaluator class internal method PartialEval. This helper simplifying tree elements that could be simplified, Expression type element has property that defines can it be simplified or not. For example, if expression tree will have some constants with operators like ToUpper that changing the case of string or substring that takes part of the constant string provided these things can be evaluated before translation. This method was taken from the article [43] series that was mentioned in section 3.1.1 and modified as version of the C# language has been changed and language itself improved, so that this method is written now according to C# version 7.3 that is latest version supported by .NET Standard 2.0 framework implementation.

Once expression tree is simplified it is passed to expression visitor for translation and as the result returns instance of the internal class WqlQuery casted to interface IWqlQuery that it is implementing.

The ExpressionVisitor [40] class helps us to implement Visitor design pattern [9] and make a translation of expression tree in more correct and structured way. As the expression visit methods require tree node to be returned as the result, _objectStack static field is used, the field is described in **Error! Reference source not found.**.

4.2.3 WqlAttribute helper class

WqlAttribute is static class that helps to get required attribute from class, type or object instance. There are four overloads of method Get described in Table 12.

Table 12 Attribute Get method, and its overloads

Nr	**Method overload**	**Description**
1	TAttribute Get<TAttribute>(object obj)	This method takes one generic argument specifies type of attribute required to be returned and one parameter that is source object instance. This method is getting type out of the object instance, by using reflection method GetType and using overload number 3 to get and return required attribute
2	TAttribute Get<TAttribute, TObject>()	This method is taking two generic arguments. First is attribute required to be returned and second is the source class name. This method is taking type from second generic parameter by using typeof method and then using overload number 3 to get and return required attribute
3	TAttribute Get<TAttribute>(Type objectType)	This method takes one generic argument specifies type of attribute required to be returned and one parameter of type that specifies the source type where required attribute needs to be found. This method is taking type of the attribute class and using overload number 4 by providing 2 Type parameters and returning its result casting it to required attribute type.
4	Attribute Get(Type attributeType, Type objectType)	This is non-generic method that takes 2 Type parameters. First is type of the required attribute and second is type of source class where this attribute needs to be found. This method uses static GetCustomAttribute method from Attribute class and returning its result.

4.2.4 ToString methods

Each object that represents part of query has override ToString method that translates itself into part of WQL [7]. All child elements' ToString methods are used by ToString method of main WqlQuery element so that in case of ToString usage WQL Query will be generated and returned as string.

As all the joins and predicates are kept in ICollection (List) object they needs to be aggregated into single string, this is done using extension methods described in section 4.2.5.

4.2.5 Extension methods

There are two static classes providing extension methods. First is WqlObjectExtensions. This class provides extension methods for WqlQuery sub elements that kept in ICollection (List) object, methods described in Table 13.

Table 13 WqlQuery collection elements extension methods

Method	**Description**
public static string AggregateString(this IEnumerable<IWqlPredicate> predicateEnumerable)	This method using LINQ [1] method Aggregate [44] that helps to create new list of strings, than add each predicate by using its ToString method and then join this list into a single string using string.Join static method and " AND " separator. The joining operation result is returned as the result of Aggregate string method.
public static string AggregateString(this IEnumerable<IWqlClassObject> joinEnumerable)	This method using LINQ [1] method Aggregate [44] that helps to create new list of strings, than add each join statement by using its ToString method and then join this list into a single string using string.Join static method and Envoriment.NewLine constant as a separator. The joining operation result is returned as the result of Aggregate string method.

The second class DateTimeExtension provides one extension method for System.DateTime type objects that helps to translate it to DMTF (Distributed Management Task Force) date time string format. This method is modified version of existing method in System.Management class [45].

4.3 “VNetDev.WmiQueryableCore.WqlTranslator.Abstraction” library

This library has all the required interfaces to represent all the translator objects and its elements.

Project creation can be done same way as described in section 4.1.1.

All interfaces described in Table 14.

Table 14 IWqlTranslator interfaces description

Interface name	**Inherited interfaces**	**Description**
IWqlFactory	*none*	This interface has one method declaration that must provide IWqlQuery object by providing Expression type argument that represents expression tree.
IWqlObject	*none*	This interface has one method declaration that allows to set object value or values by taking object array params [46].
IWqlPredicate	IWqlObject	This interface is adding three method declarations to existing in IWqlObject interface. As this method represents predicate it requires implementation of the following methods: • SetLeft – to set first object to be compared, this method requires IWqlObject as an argument. • SetRight – to set second object to be compared, this method requires IWqlObject as an argument. • SetOperator – to set comparison operator. This takes operator as string.
IWqlClassObject	IWqlObject	This interface is adding two methods declaration to existing in IWqlObject interface. Interface represents one WMI class that could be used as a main IWqlObject or any joined class. This interface requires implementation of AddPredicate method that takes one IWqlPredicate argument and returns IWqlClassObject to fulfil Fluent interface [3] requirement.
IWqlQuery	IWqlClassObject	This interface is adding one method and one read-only property declaration to existing in IWqlClassObject interface. Interface represents WqlQuery object. It requires one property of type that specifies the appropriate type of query output. Also, it requires implementation of ProceedDelegates method that applies all the delegates provided by Select [42] extension method and will transform query result into awaited format.

4.4 “VNetDev.WmiQueryableCore.Cim” library

The WMI Queryable driver provides communication over WinRM [10] protocol.

Project creation can be done same way as described in section 4.1.1.

This library can be called WMI [7] Queryable framework driver. The goal of this library is to provide connectivity to WMI by using Microsoft.Management.Infrastructure [47] package, take translation object IWqlQuery, send its WQL statement to WMI backend and map the result into appropriate objects.

The main and most important class in this library is CimConnection. As any WMI Queryable framework driver, it needs to implement IWmiConnection interface described in section IWmiConnection interface.

As IWmiConnection interface inherit IQuery Provider LINQ [1] interface as well CimConnection class must implement both.

4.4.1 CimConnection object instantiation

First CimConnection class needs to be instantiated and for that purpose it has four constructor methods described in Table 15.

Table 15 CimConnection instantiation options

Nr	Constructor signature	Description
1	CimConnection(CimSession connection, string nameSpace = @"root\CIMv2")	This constructor provides possibility to use already pre-existing Microsoft.Management.Infrastructure [47] CimSession instance. In addition, it could take one more argument to specify namespace, this can be skipped as there is default Windows namespace specified as default argument value.
2	CimConnection(string computerName = "localhost", string nameSpace = @"root\CIMv2")	This constructor takes two string arguments that allow specification of hostname or ip address of the target system and WMI namespace to connect. Both arguments are optional, and default values are local computer and default Windows namespace “root\CIMv2”. This constructor uses current user credentials for connection.
3	CimConnection(string computerName, string nameSpace, CimCredential credential)	This constructor takes three arguments, first two are same as described in constructor number 2. Third argument takes pre-existing Microsoft.Management.Infrastructure [47] CimCredential objects with preconfigured settings.
4	CimConnection(string computerName, string nameSpace, PasswordAuthenticationMechanism authenticationMechanism, string domain, string userName, string password)	This constructor allows to specify everything manually, it takes six parameters: 1. computerName – ip address or hostname of the target system 2. namespace – WMI namespace 3. authenticationMechanism – PasswordAuthenticationMechanism enum value that specifies the way of authentication. This enum is available in Microsoft.Management.Infrastructure.Options namespace and required for Microsoft.Management.Infrastructure [47] CimCredential objects creation. 4. domain – domain name to authenticate. 5. username – username to authenticate. 6. password – user password for authentication. All constructor parameters are mandatory.

4.4.1.1 IQueryProvider interface implementation

Instances of type CimConnection must implement generic and non-generic CreateQuery and Execute methods that described in Table 16.

Table 16 CimConnection methods for IQueryProvider implementation

Nr	Method	Description
1	IQueryable CreateQuery(Expression expression)	Method creates an instance of WmiClassSet<object> providing WmiContext and expression and returns it as an IQueryable.
2	IQueryable<TElement> CreateQuery<TElement>(Expression expression)	Method creates an instance of WmiClassSet<TElement> providing WmiContext and expression and returns it as an IQueryable<TElement>.
3	object Execute(Expression expression)	First method is getting IWqlQuery object by usage of WqlFactory TranslateQuery method with providing of the expression tree. Then method is checking expression trees last element to detect the result object format. If it is one object instance or single value-type, then it is checking for all the requirements and getting just one instance of the object or just result count. If list of object is awaited then it is using CreateObjectInstances method, that is described in section 0, to get IEnumerable [48] of objects required and result is returned.
4	TResult Execute<TResult>(Expression expression)	This method uses method number 3 to get the result, then cast it to appropriate type and return the result.

4.4.1.2 CreateObjectInstances method

Main goal of CreateObjectInstances method is to provide enumerable object of required object instances. This method create ObjectReader<> object instance using Activator.CreateInstance [49] method. ObjectReader class is described in section 4.4.2.

CreateObjectInstances method takes three mandatory arguments that are described in Table 17.

Table 17 CreateObjectInstances method arguments

Argument	**Type**	**Description**
type	Type	This argument is required to create ObjectReader instance, as this object requires generic argument to be specified.
queryObject	IWqlQuery	The IWqlQuery object that is the result of translation, this is required for object reader to understand WMI result object structure by using its type and alias map and then the result should be transformed, if Select method was used, to return objects in appropriate format.
instances	IEnumerable<CimInstance>	WMI object instances that are received from WMI backend.

4.4.1.3 InvokeCimMethod internal method

This is helper method that executes WMI methods using three arguments described in Table 18 and one generic argument T for result casting.

Table 18 InvokeCimMethod arguments

Argument	**Type**	**Description**
wmiClass	object	WMI object instance that represents source of the method that needs to be executed.
methodName	string	Method name that needs to be executed.
methodParameters	IDictionary<string, object>	Dictionary of parameters needs to be passed to method in order to be executed.

This method CimParameterCollection object needed with the parameters we have in Dictionary methodParameters, checking that we have requested object instance and passing them with methodName paraneter to InvokeMethod [50] method using connection we have. The result of InvokeMethod execution is translated to generic type T and rreturned as the result.

4.4.2 Object reader

ObjectReader class in needed to instantiate C# classes based on received WMI result instance and register them. This class implements IEnumerable<T> [48] that allows this object to be enumerated later.

It is creating Activator.CreateInstance [49] method to create object of IWqlQuery output type and then map all values for properties that exists in the result object instance.

Then if class has delegate methods needs to be assigned object reader will create lambda expressions that in turn uses InvokeCimMethod method in CimConnection described in section 0, compile it and assign as a value to appropriate delegate method in created object instance.

As the ObjectReader class implements IEnumerable interface it is creating object instances one by one during its further enumeration, so if object will not be used or just one or several result objects will be taken then only required objects will be created and mapped.

All the created object instances are registered in CimConnection object cache dictionary to be tracked. This cache is used when object needs any modification or needs to be used for method execution.

4.5 “VNetDev.WmiQueryableCore.DCom” library

The WMI Queryable driver provides communication over DOM [11] protocol.

Project creation can be done same way as described in section 4.1.1.

The principle of this library is very similar to driver for working over WinRM that is described in section 4.4, but the difference is that this library is using System.Management package to create a connection to WMI backend.

This is quite old way to connect to WMI and can be used only on Windows. Also, this protocol required a lot of ports to be used as DCom is using dynamic ports for connection establishment. But the biggest benefit of this approach is extended WQL [15] that is supported by some systems like Microsoft Endpoint Configuration Manager [51].

4.6 “VNetDev.WmiQueryableCore.CIMv2” library

The project that extends Base WMI context class by implementing all the classes provided in Windows default WMI namespace “root\CIMV2”.

Project creation can be done same way as described in section 4.1.1.

This project contains CIMv2WmiContext class that extends WmiContext, described in section 4.1.2. This class represents Windows default WMI namespace “root\CIMV2” that has several hundred WMI classes.

This project contains all the needed classes with all properties and method declarations. All these classes are added to the CIMv2WmiContext class via the WmiClassSet, that is described in section 4.1.4.

There are 776 classes that represent different WMI classes of the “root\CIMV2” WMI namespace, all these classes were generated by PowerShell script that is gathering all the classes from the namespace, then, by using .NET StringBuilder [52], recursively generating C# class files for each WMI class available in namespace.

5 Created solution analysis

As the result WMI Queryable framework can make developers live easier as it is making work with WMI backends a lot easier and safer as it is strongly typed and will not allow making of many mistakes that is standard weakly typed approach can be made quite often.

5.1 Comparison of old and new ways of working

With help of new WMI Queryable framework code can be shorter and more readable than with old ways of working that is still standard way of doing WMI Queries. The difference between code blocks is represented on Figure 11 and Figure 12.

```
var computer = "Computer_B";
var domain = "domain.local";
var username = "AdminUserName";
var password = "Password123";

SecureString securePassword = new SecureString();

foreach (var c in password)
{
    securePassword.AppendChar(c);
}

var credentials = new CimCredential(
    PasswordAuthenticationMechanism.Default,
    domain,
    username,
    securePassword);

var sessionOptions = new WSManSessionOptions();

sessionOptions.AddDestinationCredentials(credentials);

var session = CimSession.Create(
    computer, sessionOptions);

var allVolumes = session
    .QueryInstances(
        @"root\cimv2",
        "WQL",
        "SELECT * FROM Win32_Volume WHERE DriveLetter <> NULL");

foreach (var oneVolume in allVolumes)
{
    Console.WriteLine(
        "Volume '{0}' has {1} bytes total, {2} bytes available",
        oneVolume.CimInstanceProperties["DriveLetter"],
        oneVolume.CimInstanceProperties["Size"],
        oneVolume.CimInstanceProperties["SizeRemaining"]);
}
```

Figure 11 Example of old way of working.

```
var context = new CIMv2WmiContext();
context.Configure(x => x.UseCim(
    "Computer_B", "root\\CIMV2", PasswordAuthenticationMechanism.Default,
    "domain.local", "AdminUserName", "Password123"));

foreach (var volume in context.Win32Volume
    .Where(x => x.DriveLetter != null))
    Console.WriteLine(
        $"Volume '{volume.DriveLetter}' has {volume.Capacity}" +
        " bytes total, {volume.FreeSpace} bytes available");
```

Figure 12 Example of new way of working

5.2 Possibilities for further improvements

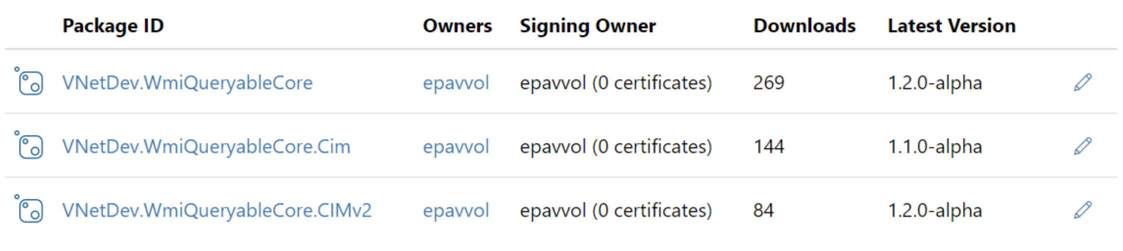
There is some room for improvement of this WMI Queryable framework available.

Currently it does not support asynchronous querying, this for sure will be very beneficial thing to implement.

DCom driver could be improved to support Linux OS as well as Windows. This could be done using command line tools available for WMI on Linux.

5.3 Unexpected statistics

During development process author has released test alpha version of the WMI Queryable frameworks and published them to Nuget.org [4] package repository for its own use. During three days' time period there already were several hundred downloads, statistics shown on Figure 13.



Package ID	Owners	Signing Owner	Downloads	Latest Version
VNetDev.WmiQueryableCore	epavvol	epavvol (0 certificates)	269	1.2.0-alpha
VNetDev.WmiQueryableCore.Cim	epavvol	epavvol (0 certificates)	144	1.1.0-alpha
VNetDev.WmiQueryableCore.CIMv2	epavvol	epavvol (0 certificates)	84	1.2.0-alpha

Figure 13 Nuget.org packages download statistics

5.4 Further development

Next actions will be finalizing and testing of the created solution. Solution will be uploaded to appropriate GitHub repository and new versions of WMI Queryable interface will be created and pushed to Nuget.org package repository.

Packages will be available in “Package Manager Console” in Microsoft Visual Studio or JetBrains Rider IDE. Also, these packages can be downloaded from the nuget.org website.

List of packages available:

1. VNetDev.WmiQueryableCore
2. VNetDev.WmiQueryableCore.WqlTranslator
3. VNetDev.WmiQueryableCore.WqlTranslator.Abstraction
4. VNetDev.WmiQueryableCore.Cim
5. VNetDev.WmiQueryableCore.DCom
6. VNetDev.WmiQueryableCore.CIMv2

GitHub repository available at the following URL:

https://github.com/epavvol/WmiQueryableCore

6 Summary

This thesis deals with inconvenience in interaction with Windows Management Instrumentation [5] in .NET [6] environment. As described in chapter 1 there are no good tools in any of the .NET framework implementations that help developers to interact with WMI in convenient, safe and readable way. All approaches that allows interaction with WMI require a lot of effort to prepare required mechanism and objects. This way of development has a lot of drawbacks and places of potential mistakes that will pop up only in runtime and this makes development and debugging processes a lot more difficult and as the result code is less readable, that makes further improvements a lot more problematic.

In the chapter 2 author pointed out that there are still no good solutions that could solve all the issues with WMI interaction in .NET environment, on the contrary, there are a lot of forum topics that prove the relevance of the problem, where other developers are looking for the solution of the same problem.

In the same chapter author is suggesting possible way of solving these issues by standardizing principles of working with WMI in .NET environment. The idea is to make possibility of working in single WMI context object with classes that have strongly typed properties. This approach makes development process safer and eliminates a lot of potential mistakes, as biggest part of logical mistakes become syntax errors and, as the result, compiler will force developer to fix them.

Chapter 3 describes analysis part of the current thesis. In this chapter the author explores the possibilities that are best suited to solve the problem in the best way. After long research the author concludes that the best suitable technology, which is available in .NET environment, is LINQ [1] IQueryable [2] interface, that provides very convenient fluent interface [3].

Same chapter describes several available .NET framework implementations and the decision that author considers more suitable is to use .NET Standard framework version

2.0 as this version is supported almost in all .NET implementations and .NET based languages [27].

In the chapter 4 author describes solution implementation process in more detailed level. This chapter shows all the techniques used during development process and describes all implemented design patterns and principles. In this section author is explaining all the objects, their purposes and relationships between them.

The 5th chapter shows the analysis of the solution created. The comparison that the author cites in his work proves that code blocks are reduced a lot and now they are written in more readable and safe way than without usage of WMI Queryable framework.

In the same chapter the author shared his thoughts on further development.

In author's opinion the prototype of WMI Queryable is successful and will be very useful in .NET development of applications that require interaction with Windows Management Instrumentation. The framework created will be improved in future and supported in case of demand.

References

[1] "Language Integrated Query (LINQ)," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/. [Accessed 25 11 2019].

[2] "IQueryable Interface," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.linq.iqueryable?view=netstandard-2.0. [Accessed 25 11 2019].

[3] "Fluent interface," [Online]. Available: https://en.wikipedia.org/wiki/Fluent_interface. [Accessed 17 11 2019].

[4] "NuGet," [Online]. Available: https://www.nuget.org/. [Accessed 10 10 2019].

[5] "Windows Management Instrumentation," Microsoft, 31 05 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page. [Accessed 15 11 2019].

[6] ".NET documentation," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/. [Accessed 15 12 2019].

[7] "WQL (SQL for WMI)," Microsoft, 31 05 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wql-sql-for-wmi. [Accessed 05 11 2019].

[8] "Query Syntax and Method Syntax in LINQ (C#)," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq. [Accessed 25 11 2019].

[9] "Visitor pattern," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Visitor_pattern. [Accessed 07 01 2020].

[10] "Windows Remote Management," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/winrm/portal. [Accessed 05 11 2019].

[11] "[MS-DCOM]: Distributed Component Object Model (DCOM) Remote Protocol," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/4a893f3d-bd29-48cd-9f43-d9777a4415b0. [Accessed 15 02 2020].

[12] "SOLID," [Online]. Available: https://en.wikipedia.org/wiki/SOLID. [Accessed 20 11 2019].

[13] "Application programming interface," [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface. [Accessed 15 02 2020].

[14] "GitHub," [Online]. Available: https://github.com/. [Accessed 25 10 2019].

[15] "Configuration Manager Extended WMI Query Language," Microsoft, 20 09 2016. [Online]. Available: https://docs.microsoft.com/en-

us/mem/configmgr/develop/core/understand/extended-wmi-query-language. [Accessed 25 11 2019].

[16] "ASP.NET Core," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1. [Accessed 15 12 2019].

[17] "Dependency injection in ASP.NET Core," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1. [Accessed 20 12 2019].

[18] "LINQ: BUILDING AN IQUERYABLE PROVIDER SERIES," [Online]. Available: https://web.archive.org/web/20090210162932/http://blogs.msdn.com/mattwar/pages/linq-links.aspx. [Accessed 25 11 2019].

[19] "SQL," [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed 05 11 2019].

[20] "Entity Framework documentation," [Online]. Available: https://docs.microsoft.com/en-us/ef/. [Accessed 10 11 2019].

[21] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns," in *Design Patterns: Elements of Reusable Object-Oriented Software,* Pearson Education, 1994, p. 395.

[22] D. Nesteruk, "Design Patterns in C# and .NET," Udemy, 04 2020. [Online]. Available: https://www.udemy.com/course/design-patterns-csharp-dotnet/. [Accessed 15 04 2020].

[23] "Connecting to WMI Remotely with C#"," [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/wmisdk/connecting-to-wmi-remotely-with-c-. [Accessed 25 10 2019].

[24] ".NET Framework," [Online]. Available: https://en.wikipedia.org/wiki/.NET_Framework. [Accessed 15 12 2019].

[25] ".NET Core," [Online]. Available: https://en.wikipedia.org/wiki/.NET_Core. [Accessed 01 02 2020].

[26] ".NET Standard," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/net-standard. [Accessed 25 11 2019].

[27] ".NET Programming Languages," Microsoft, [Online]. Available: https://dotnet.microsoft.com/languages. [Accessed 25 11 2019].

[28] "Integrated development environment," [Online]. Available: https://en.wikipedia.org/wiki/Integrated_development_environment. [Accessed 20 10 2019].

[29] "Visual Studio," Microsoft, [Online]. Available: https://visualstudio.microsoft.com/. [Accessed 05 10 2019].

[30] "ReSharper," JetBrains, [Online]. Available: https://www.jetbrains.com/resharper/. [Accessed 01 10 2019].

[31] "Rider," JetBrains, [Online]. Available: https://www.jetbrains.com/rider/. [Accessed 01 10 2019].

[32] "GitHub," [Online]. Available: https://en.wikipedia.org/wiki/GitHub. [Accessed 25 10 2019].

[33] "Bitbucket," Atlassian, [Online]. Available: https://en.wikipedia.org/wiki/Bitbucket. [Accessed 19 12 2019].

[34] “GitLab,” [Online]. Available: https://en.wikipedia.org/wiki/GitLab. [Accessed 25 10 2019].

[35] “Builder pattern,” [Online]. Available: https://en.wikipedia.org/wiki/Builder_pattern. [Accessed 20 01 2020].

[36] “IListSource Interface,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.ilistsource?view=netstandard-2.0. [Accessed 05 11 2019].

[37] “CSPROJ File Format,” [Online]. Available: https://wiki.fileformat.com/programming/csproj/. [Accessed 20 02 2020].

[38] “XML,” [Online]. Available: https://en.wikipedia.org/wiki/XML. [Accessed 20 11 2019].

[39] “Repository and Unit of Work Pattern,” 09 01 2018. [Online]. Available: https://www.programmingwithwolfgang.com/repository-and-unit-of-work-pattern/. [Accessed 19 01 2020].

[40] “ExpressionVisitor Class,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.linq.expressions.expressionvisitor?view=netcore-3.1. [Accessed 20 12 2019].

[41] “#region (C# Reference),” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/preprocessor-directives/preprocessor-region. [Accessed 05 01 2020].

[42] “Queryable.Select Method,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.linq.queryable.select?view=netcore-3.1. [Accessed 25 11 2019].

[43] “Partial Expression Evaluator,” [Online]. Available: https://web.archive.org/web/20090221134246/http://blogs.msdn.com/mattwar/archive/2007/08/01/linq-building-an-iqueryable-provider-part-iii.aspx. [Accessed 25 11 2019].

[44] “Enumerable.Aggregate Method,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.aggregate?view=netcore-3.1. [Accessed 20 01 2020].

[45] “System.Management source,” [Online]. Available: https://github.com/dotnet/runtime/blob/ccf6aedb63c37ea8e10e4f5b5d9d23a69bdd9489/src/libraries/System.Management/src/System/Management/ManagementDateTime.cs. [Accessed 25 11 2019].

[46] “params (C# Reference),” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/params. [Accessed 15 01 2020].

[47] “Microsoft.Management.Infrastructure,” [Online]. Available: https://github.com/PowerShell/MMI. [Accessed 25 11 2019].

[48] “IEnumerable Interface,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=netcore-3.1. [Accessed 05 11 2019].

[49] “Activator.CreateInstance Method,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.activator.createinstance?view=netcore-3.1. [Accessed 05 03 2020].

[50] "CimSession.InvokeMethod Method," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/microsoft.management.infrastructure.cimsession.invokemethod?view=pscore-6.2.0. [Accessed 15 02 2020].

[51] "Microsoft Endpoint Configuration Manager," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/mem/configmgr/. [Accessed 10 01 2020].

[52] "StringBuilder Class," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.text.stringbuilder?view=netcore-3.1. [Accessed 10 12 2019].