**Logging Guidelines**

Atul Badgujar

05-April-2020

Version: 1.0

Document Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Revision | Revision By | Review By | Approval By |
|  |  |  |  |  |
|  |  |  |  |  |

This document is currently WIP. It will be updated frequently for some more time before it becomes complete.

# Introduction

This document lists and explains guidelines for using logging statements in a .Net Core application. It contains many examples of logging.

# Serilog

We will use Serilog as our logger. See https://serilog.net/

It can be also installed as a middleware.   
Installation: <https://github.com/serilog/serilog/wiki/Getting-Started>

Also install following plugins/addons:

Install-Package Serilog

Install-Package Serilog.Aspnetcore

Install-Package Serilog.Sinks.Console  
Install-Package Serilog.Sinks.RollingFiles  
Install-Package Serilog.Sinks.Seq

For Thread properties: install

[Install-Package Serilog.Enricher.Thread](https://github.com/serilog/serilog-enrichers-thread)

After installing thread Enricher Package, add the Enricher to the Logger Configuration, this will add Thread Id property to the Logs and the logs can be filtered with Thread Id.

.Enrich.WithThreadId()

# Setup and Initialization

See following example:

For Development Environment:

public class Program {

/// <summary>

/// Initializes Webhost along with Serilog

/// Logs to File

/// Logs to console

/// logs to Seq

/// Logs in json format in a file

/// Adds Property - Context, ThreadId,Application Name, Version

/// Also for request(Middleware) - sessionId,UserId,TenantId,

/// Elapsedtime and request

/// </summary>

/// <param name="args"></param>

public static void Main(string[] args) {

//Read Appsettings

var configuration = new ConfigurationBuilder()

.AddJsonFile("AppSettings/appsettings.json")

.Build();

string appName = configuration.GetValue<string>("AppName");

string appVersion = configuration.GetValue<string>("AppVersion");

//Define Serilog Logger model

LoggerModel model = new LoggerModel { AppName = appName, AppVersion =

appVersion, SeqURL = "http://localhost:5341" };

//Get serilog logger and insert to the Static Log class

Log.Logger = SerilogLogger.Configure(model,null);

try {

Log.Information("Starting web host");

CreateWebHostBuilder(args).Build().Run();

}

catch (Exception ex) {

Log.Fatal(ex, "Host terminated unexpectedly");

}

finally {

//As logger may be using some heavy resources

//it must be disposed on exit

Log.CloseAndFlush();

}

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

// this replaces default logger with Serilog logging

// As serilog uses older version of ASp.netcore.app library

// this function can not be shifted to the dll.

.UseSerilog();

}

Note: Model has Boolean for all type of sinks like Email,Seq,Console and Rollingfile, If these sinks are no required set the value false for the given sink.

Also Serilog settings can be read by the Configuration file also and below is the example, how it can be configured by Appsetting, but we are not using it for brevity.

Example - In app settings add Serilog section, it read from that section-

appsettings.jsonConfiguraion - Serilog.Settings.Configuration

{

"Serilog": {

"Using": ["Serilog.Sinks.Console"],

"MinimumLevel": "Debug",

"WriteTo": [

{ "Name": "Console" },

{ "Name": "File", "Args": { "path": "%TEMP%\\Logs\\serilog-

configuration-sample.txt" } }

],

"Enrich": ["FromLogContext", "WithMachineName", "WithThreadId"],

"Destructure": [

{ "Name": "With", "Args":{ "policy": "Sample.CustomPolicy, Sample" } },

{ "Name": "ToMaximumDepth", "Args":{"maximumDestructuringDepth": 4 } },

{ "Name": "ToMaximumStringLength","Args":{"maximumStringLength":100} },

{ "Name": "ToMaximumCollectionCount","Args":{"maximumCollectionCount": 10 } }

],

"Properties": {

"Application": "Sample",

“ApplicationVersion” : “1.0.0”

}

}

}

Make following changes to Program.cs to use the config file for serilog configuration -

public class Program

{

public static void Main(string[] args)

{

var configuration = new ConfigurationBuilder()

.AddJsonFile("appsettings.json")

.Build();

var logger = new LoggerConfiguration()

.ReadFrom.Configuration(configuration)

.CreateLogger();

logger.Information("Hello, world!");

}

}

# Ad-hoc Logger

Sometimes during debugging, we may need a separate logger for a short time. Following is an example of it:

Serilog.Ilogger customLogger = new LoggerConfiguration()  
 .WriteTo.Console()

.WriteTo.File("log.txt"))

.CreateLogger();

Anything logged by using custom Logger will use the configuration given for this logger, like it will log to the log.txt file.

# Email Logging

When necessary, Serilog can be used to send a log statement via email by using an ad-hoc Email Logger.

Serilog.Ilogger emailLogger =

SerilogLoggerService.GetEmailLogger(EmailLoggerModel model)

EmailLogger.LogError(“test”);

Function Implementation Detail:

SerilogLoggerService.GetEmailLogger(EmailLoggerModel model){

ILogger logger = new LoggerConfiguration()

.MinimumLevel.Debug()

.MinimumLevel.Override("Microsoft", LogEventLevel.Information)

.Enrich.FromLogContext()

.Enrich.WithThreadId()

.Enrich.WithProperty("Application", model.AppName)

.Enrich.WithProperty("ApplicationVersion", model.AppVersion)

.WriteTo.Email(new EmailConnectionInfo {

FromEmail = model.SenderEmail,

ToEmail = model.ReceiverEmail,

MailServer = model.EmailServer, //"smtp.gmail.com",

NetworkCredentials = new NetworkCredential {

UserName = model.UserName,

Password = model.Password

},

EnableSsl = true,

Port = 465,

EmailSubject = model.EmailSubject

},

outputTemplate: "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}]

{Message}{NewLine}{Exception}",

batchPostingLimit: 10

, restrictedToMinimumLevel: Serilog.Events.LogEventLevel.Error

)

.CreateLogger();

Return logger;

}

# SEQ Sink

The beauty of Serilog logger is that it supports various database sinks that can be used to search appropriate log statements in flexible and sophisticated ways. SEQ sink is one of them.

## Seq Installation:

Install Package Serilog.Sinks.Seq  
Install MSI from <https://getseq.net/>

## Seq Configuration:

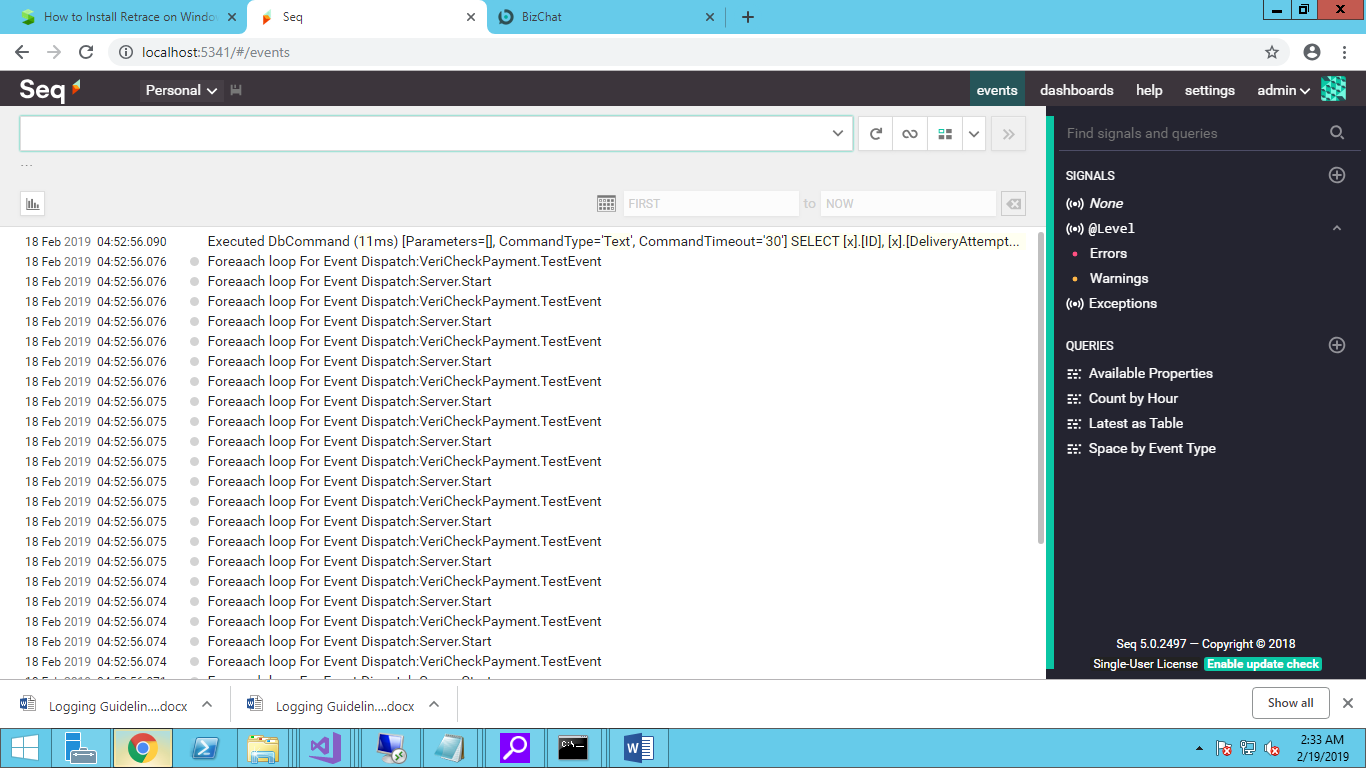
The Serilog logger can write log to Seq which can be used to View, Filter the log statements. Seq can be configured to write by Serilog by following statement and can be used to view the logs on URL given in the configuration

Note: SEQ MSI should be installed on given URL.

.WriteTo.Seq("http://localhost:5341")

## Help on Seq Usage and Examples:

<http://docs.getseq.net/docs>



# Retrace and Stackify

Asha/Hari: Later

## Retrace sink

a. Add Nuget Serilog.Sinks.Stickify

b. Add following setting to App Settings -

<appSettings>

<add key="Stackify.ApiKey" value="Your Activation Key" />

<add key="Stackify.AppName" value="Your App Name"/>

<!-- optional - will be inferred by Stackify if this is running on a

monitored server -->

<add key="Stackify.Environment" value="Your Environment"/>

<!-- optional - will be inferred by Stackify if this is running on a

monitored server -->

</appSettings>

c. Add the log output for stackify by following code

.WriteTo.Stackify()

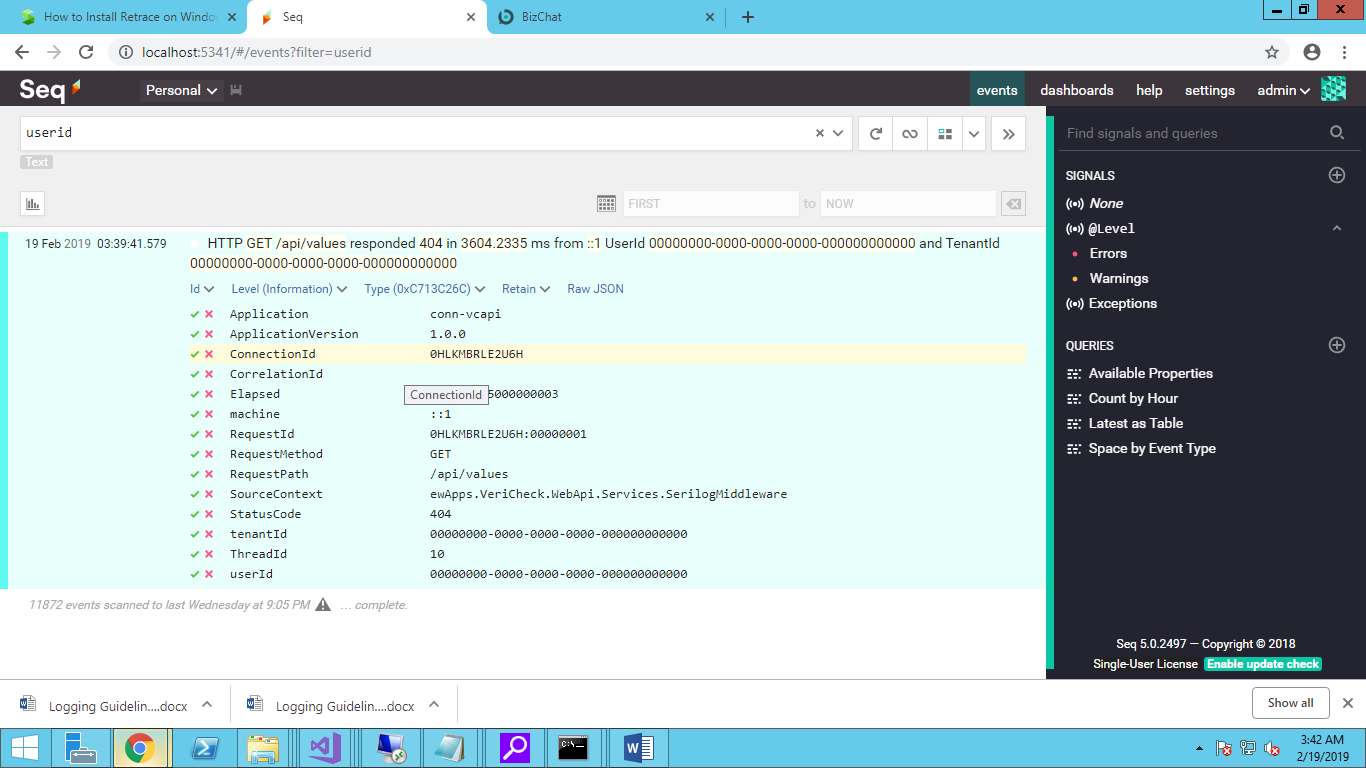
# Serilog Middleware

All the request can be logged for the Time request took to execute the request Id, or called IP address and Method it is called, Session Info etc., by the Serilog Middleware.

app.UseMiddleware<SerilogMiddleware>();

Note: If Session is used then in Configuration write the middleware after Session Middleware.

# Serilog Middleware log example on Seq URL



# Serilog Timing and Metrics

See: <https://github.com/nblumhardt/serilog-timings>

It describes what the package does and how to use it.

# Guidelines

1. Do not use any other logging tools.
2. Configure different logging environments for (1) Development and (2) Production.
3. Use logging levels accurately. This is IMPORTANT.
4. Format and indent logging statements as per our style.
5. In Development environment, temporary logging statements may be added, updated and deleted frequently. Make certain that these statements are deleted after their use.
6. In Visual Studio environment, log statements should be output to “Output” window.

# Notes

1. To include the ClassName in the log, see below.  
   ForContext example – Using the ILogger<ClassName> creates the Context  
   SourceType (ClassName) enricher: .Enrich.FromLogContext())
2. MessageTemplate – With Serilog, log output can be written to multiple output , Like if we want to log the statements in a file with given format and also it has its own minimum logging level, following code can be used to define to use file output with given template for output and minimum log level will be error to log in this file.  
     
   .WriteTo.File("log.txt", outputTemplate:   
    "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj} {NewLine} {Exception}", restrictedToMinimumLevel: Serilog.Events.LogEventLevel.Error)
3. Customize output format   
   .WriteTo.File("log.txt", outputTemplate:   
    "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj} {NewLine} {Exception}", restrictedToMinimumLevel: Serilog.Events.LogEventLevel.Error)

# To Do

1. Azure sink
2. LogContext - Push/Pop example

# Useful Links

1. Comparison:   
   <https://stackify.com/nlog-vs-log4net-vs-serilog/>
2. Tutorial (\*Good\*):   
   <https://blog.getseq.net/serilog-tutorial/>  
   <https://stackify.com/serilog-tutorial-net-logging/>
3. Structured Logging: <https://stackify.com/what-is-structured-logging-and-why-developers-need-it/>
4. Retrace: <https://stackify.com/retrace/>
5. ASP.Net  
   <https://andrewlock.net/adding-serilog-to-the-asp-net-core-generic-host/>
6. As middleware: <https://blog.getseq.net/smart-logging-middleware-for-asp-net-core/>   
   WebClassic (Exception and request timing): <https://github.com/serilog-web/classic>
7. .Net Core Enricher libraries: <https://github.com/serilog/serilog-aspnetcore>