**Logging Framework**

Contents

[1 Document Control 3](#_Toc98937191)

[1.1 Change Record 3](#_Toc98937192)

[1.2 Reviewer 3](#_Toc98937193)

[1.3 Approver 3](#_Toc98937194)

[2 Document Purpose 3](#_Toc98937195)

[3 Logging Frameworks 4](#_Toc98937196)

[3.1 Log4net 4](#_Toc98937197)

[3.2 NLog 4](#_Toc98937198)

[3.3 Serilog 5](#_Toc98937199)

[3.3.1 Structured logging example 5](#_Toc98937200)

[3.4 Microsoft Extension Logging 6](#_Toc98937201)

[4 Comparison of Frameworks 6](#_Toc98937202)

[5 Conclusion 7](#_Toc98937204)

[6 Best Practises 7](#_Toc98937206)

[6.1.1 What should log 8](#_Toc98937207)

[6.1.2 What shouldn’t log 8](#_Toc98937208)

[6.1.3 Log Types 8](#_Toc98937209)

[6.1.4 Log Levels 8](#_Toc98937210)

[6.1.5 Log Context 9](#_Toc98937211)

[6.1.6 Concurrency in logs 10](#_Toc98937212)

[6.1.7 Use structured logs 10](#_Toc98937213)

[6.1.8 General Rules 11](#_Toc98937215)

[7 Appendix 12](#_Toc98937216)

# Document Control

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change reference** |
| 12/17/2021 | Prajeesh T S | 1.1 | Initial version |

## Reviewer

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

## Approver

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

# Document Purpose

This document provides details regarding the logging frameworks and the best practices

# Logging Frameworks

## Log4net

log4net has been used in tens of thousands of applications over the last 17 years; certainly, it is the grandparent of all modern .NET logging frameworks. Concepts such as log levels, loggers are nearly universal in logging frameworks.

This contains the XML configuration file that contains the settings for logging. Don’t forget to set that file to copy to the output directory.

It is complicated and XML is always difficult to approach.

log4net hasn’t seen a release in 18 months, which isn’t necessarily bad as the project is stable.



## NLog

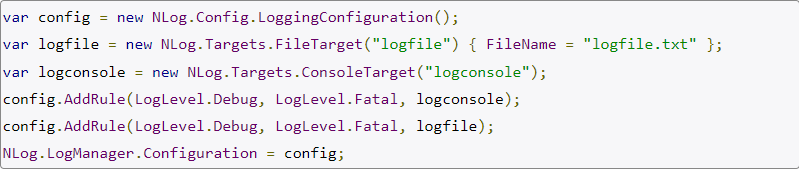
NLog is also a pretty old project. Version 1.0 was released back in 2006, but it is still under active development with the latest version having been released in December of 2017

The latest release of NLog adds structured logging and support for .NET Standard.

This also has XML configuration file and it is cleaner than log4net.



There is also a way to configure the logging using code.



## Serilog

The newest logging framework of the bunch, Serilog, was released in 2013. The big difference between Serilog and the other frameworks is that it is designed to do structured logging out of the box. NLog also supports structured logging but it is was only added recently.

### Structured logging example

Often, you’ll find that you’re writing logs that contain two things: a message and value.

Example:

log.Debug($"User id is: ${userId}");

With most logging frameworks, this is simply translated to text in the log file. Text is nice and all, but knowing that the value logged was called userId and being able to search for that is actually very useful. Serilog keeps the properties available all the way to the destination.

No need for any XML configuration this time! The configuration for Serilog uses a fluent interface, which makes it very nice and clean. Compare how easy it is to set up a rolling file logger, as compared with log4net.

## Microsoft Extension Logging

.NET Core and ASP.NET Core come with its own built-in logging framework. When you create a new ASP.NET Core project from the default template, Microsoft Extension Logging will already be included in the project.

ASP.NET Core Logging framework is both an abstraction and implementation. It primarily makes sure you can get the ILogger<T> interface in your ASP.NET Core dependency injection system. You’ll be able to do the following in your controllers:

# Comparison of Frameworks

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Log4net** | **Nlog** | **Serilog** |
| License | Apache License 2.0 | BSD  Open Source | Apache License 2.0 |
| Configuration | Requires XML file | Requires XML file | No Configuration [XML] files |
| Structured Logging | No | Yes [Recently added] | Yes |
| Last Update | 2 years ago, | May 30 2020 | Few months back |
| Easily Configurable | Difficulty in configuration | The config file is fairly easy to understand and modify without having to pore over pages and pages of documentation. | Flexible configuration options - Configuring directly or using app settings is fairly and there are many examples available. |
| Documentation | Documentation is quite difficult to navigate through unless you know exactly what you're looking for. | Good | Good |
| Easy to setup | Little difficult due to configuration files and difficult documentation | Getting starting with NLog is quick and you can be up and running in a matter of minutes by following the official Getting Started tutorial. | Easy |
| Portable | No | Not able to configure it for Xamarin Platform. | Portable between recent .NET platforms |
| Community | Less [ Not updated recently] | Good | Great Community |
| Total Downloads | 1975222 | 2470448 | 5474291 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Conclusion

# Seri logs API is more modern, it is easier to set up, it is better maintained and it does structured logging by default. The ability to add enrichers give the ability to intercept and modify the messages is useful.

# Best Practises

### What should log

* + All errors and exceptions (Including stack traces)
  + Anything related to security and audit (who did what)
  + Action taken by users and their result (Audit logs)

### What shouldn’t log

* + Sensitive data (Service passwords, token etc)
  + Users private data (Passwords, tokens, financial data)
  + Personal identifiable information (personal name.)

### Log Types

* Development Logs
  + Trace
  + Debug
* General Logs
  + Info
* Error Logs
  + Warning
  + Error
  + Fatal
* Audit Logs
  + Actions of users

### Log Levels

|  |  |  |  |
| --- | --- | --- | --- |
| Log Level | Value | Method | Description |
| Trace | 0 | LogTrace | Contain the most detailed messages. These messages may contain sensitive app data. These messages are disabled by default and should ***not*** be enabled in production. |
| Debug | 1 | LogDebug | For debugging and development. Use with caution in production due to the high volume. |
| Information | 2 | LogInformation | Tracks the general flow of the app. May have long-term value. |
| Warning | 3 | LogWarning | For abnormal or unexpected events. Typically includes errors or conditions that don't cause the app to fail. |
| Error | 4 | LogError | For errors and exceptions that cannot be handled. These messages indicate a failure in the current operation or request, not an app-wide failure. |
| Critical | 5 | LogCritical | For failures that require immediate attention. Examples: data loss scenarios, out of disk space. |

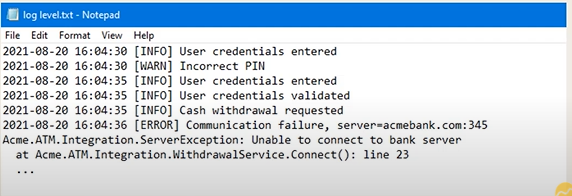
### Log Context

The log should have proper information about the context.

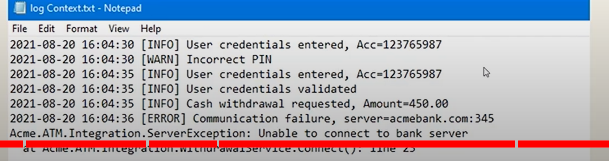
Include the context information’s like

* User (User Id, User name etc)
* The current state
* Parameters of the action
* Result of previous actions

Log without context information’s

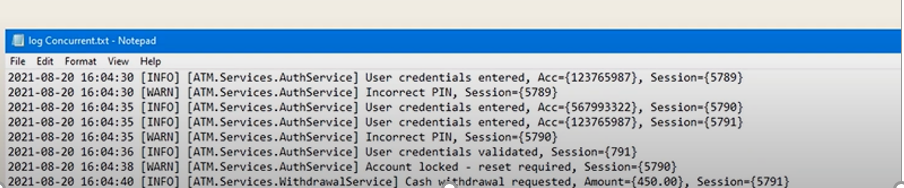


Log with proper context information’s



### Concurrency in logs

Use trace Id or correlation id to distinguish each request, use middle wares to append correlation id or trace id in request/response headers.



### Use structured logs

### It is recommended to always use semantic/structured logs so that the logging backend receives the string with placeholders and its values separately.



Advantages of structured logs:

* Properties are stored as separate custom properties of the log entry and you can filter based on them in the logging backend
* A message template/message hash is stored so that you can easily query for all logs of a given log statement type
* Serialization of properties only happens if the log is actually written (not the case with string interpolation)

### General Rules

#### Always pass exception as first parameter

To log an exception, always pass the exception object as first argument: 

#### Always use place holders and avoid string interpolation

To ensure that a message template is correctly logged and properties are transferred as custom properties, you always need to log properties with placeholders in the correct order, e.g.



#### Do not use dots in property names

Avoid using dots in placeholder property names (e.g Device.ID) because some ILogger implementations (e.g. Serilog) do not support this.

#### Use the Warning and Information Log Levels Judiciously

Warning should be reserved for logs that are not errors—they don’t impede execution—but are not entirely normal behavior. Examples might be a cache miss on an expensive object that *really* should have been in the cache, a piece of code that completed but took longer than expected

Information should be used when the application is executing normally but you want to communicate something to the operator. The most common of these might be application life-cycle events, like Application started or Application stopped.

#### Never Log Personal Identifying Information or Secrets

PII you should avoid logging includes, but is not limited to:

* First Name
* Last Name
* Username
* Gender
* Birthday
* Mailing/Billing Addresses
* Email Addresses
* Phone Numbers
* Social Security Number
* Credit Card Numbers

# Appendix

* [Logging in .NET | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/core/extensions/logging?tabs=command-line#log-level)
* [Logging with ILogger in .NET: Recommendations and best practices - Rico Suter's blog. (rsuter.com)](https://blog.rsuter.com/logging-with-ilogger-recommendations-and-best-practices/)