# Advanced Serilog Configuration in .NET 8 Class Library

Here's a comprehensive implementation of Serilog configuration for a .NET 8 class library that includes:

A custom ILogEventEnricher implementation

A static logging configuration class

Performance logging extensions

Capture of class name, method name, line numbers, and execution time

## 1. Custom Log Event Enricher

csharp

Copy

Download

using Serilog.Core;using Serilog.Events;

namespace YourNamespace.Logging{

public class ApplicationDetailsEnricher : ILogEventEnricher

{

public void Enrich(LogEvent logEvent, ILogEventPropertyFactory propertyFactory)

{

var assemblyName = typeof(ApplicationDetailsEnricher).Assembly.GetName().Name;

var assemblyVersion = typeof(ApplicationDetailsEnricher).Assembly.GetName().Version?.ToString() ?? "unknown";

logEvent.AddPropertyIfAbsent(propertyFactory.CreateProperty("ApplicationName", assemblyName));

logEvent.AddPropertyIfAbsent(propertyFactory.CreateProperty("ApplicationVersion", assemblyVersion));

logEvent.AddPropertyIfAbsent(propertyFactory.CreateProperty("EnvironmentName", Environment.GetEnvironmentVariable("ASPNETCORE\_ENVIRONMENT") ?? "Unknown"));

}

}}

## 2. Static Logging Configuration Class

csharp

Copy

Download

using Serilog;using Serilog.Exceptions;using Serilog.Formatting.Compact;using Serilog.Formatting.Json;

namespace YourNamespace.Logging{

public static class LoggingConfiguration

{

public static ILogger CreateLogger(IConfiguration configuration = null)

{

var loggerConfig = new LoggerConfiguration()

.MinimumLevel.Debug()

.Enrich.FromLogContext()

.Enrich.WithExceptionDetails()

.Enrich.WithMachineName()

.Enrich.WithProcessId()

.Enrich.WithThreadId()

.Enrich.With<ApplicationDetailsEnricher>();

if (configuration != null)

{

loggerConfig.ReadFrom.Configuration(configuration);

}

else

{

// Default configuration if no external config is provided

loggerConfig

.WriteTo.Console(outputTemplate: "[{Timestamp:HH:mm:ss} {Level:u3}] {Message:lj} {NewLine}{Exception}")

.WriteTo.File(

new CompactJsonFormatter(),

"logs/log-.json",

rollingInterval: RollingInterval.Day,

retainedFileCountLimit: 7);

}

return loggerConfig.CreateLogger();

}

}}

## 3. Performance Logging Extensions

csharp

Copy

Download

using System.Diagnostics;using System.Runtime.CompilerServices;using Serilog;

namespace YourNamespace.Logging{

public static class PerformanceLoggingExtensions

{

public static ILogger Here(

this ILogger logger,

[CallerMemberName] string memberName = "",

[CallerFilePath] string sourceFilePath = "",

[CallerLineNumber] int sourceLineNumber = 0)

{

return logger

.ForContext("MemberName", memberName)

.ForContext("ClassName", Path.GetFileNameWithoutExtension(sourceFilePath))

.ForContext("SourceFile", sourceFilePath)

.ForContext("LineNumber", sourceLineNumber);

}

public static IDisposable TimeOperation(

this ILogger logger,

string operationName,

[CallerMemberName] string memberName = "",

[CallerFilePath] string sourceFilePath = "",

[CallerLineNumber] int sourceLineNumber = 0)

{

var sw = Stopwatch.StartNew();

return new DisposableStopwatch(logger, sw, operationName,

memberName, Path.GetFileNameWithoutExtension(sourceFilePath),

sourceLineNumber);

}

private class DisposableStopwatch : IDisposable

{

private readonly ILogger \_logger;

private readonly Stopwatch \_stopwatch;

private readonly string \_operationName;

private readonly string \_memberName;

private readonly string \_className;

private readonly int \_lineNumber;

public DisposableStopwatch(

ILogger logger,

Stopwatch stopwatch,

string operationName,

string memberName,

string className,

int lineNumber)

{

\_logger = logger;

\_stopwatch = stopwatch;

\_operationName = operationName;

\_memberName = memberName;

\_className = className;

\_lineNumber = lineNumber;

}

public void Dispose()

{

\_stopwatch.Stop();

\_logger.Here(\_memberName, $"{\_className}.cs", \_lineNumber)

.Information("Operation {OperationName} completed in {ElapsedMilliseconds}ms",

\_operationName, \_stopwatch.ElapsedMilliseconds);

}

}

}}

## 4. Exception Logging Extension

csharp

Copy

Download

using System.Runtime.CompilerServices;using Serilog;

namespace YourNamespace.Logging{

public static class ExceptionLoggingExtensions

{

public static void LogException(

this ILogger logger,

Exception ex,

string message = "An error occurred",

[CallerMemberName] string memberName = "",

[CallerFilePath] string sourceFilePath = "",

[CallerLineNumber] int sourceLineNumber = 0)

{

logger.Here(memberName, sourceFilePath, sourceLineNumber)

.Error(ex, message);

}

}}

## Usage Examples

### Basic Logging

csharp

Copy

Download

public class SampleService{

private readonly ILogger \_logger;

public SampleService(ILogger logger)

{

\_logger = logger.ForContext<SampleService>();

}

public void DoWork()

{

\_logger.Here().Information("Starting work");

try

{

using (\_logger.TimeOperation("ComplexCalculation"))

{

// Perform some work

Thread.Sleep(100);

}

}

catch (Exception ex)

{

\_logger.LogException(ex, "Failed to complete work");

throw;

}

}}

### Initialization in .NET 8

csharp

Copy

Download

// Program.cs or startup classvar builder = WebApplication.CreateBuilder(args);

// Configure Serilog

builder.Host.UseSerilog(LoggingConfiguration.CreateLogger(builder.Configuration));

// Register services

builder.Services.AddSingleton<SampleService>();

var app = builder.Build();

## Configuration in appsettings.json

json

Copy

Download

{

"Serilog": {

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

"MinimumLevel": {

"Default": "Information",

"Override": {

"Microsoft": "Warning",

"System": "Warning"

}

},

"WriteTo": [

{

"Name": "Console",

"Args": {

"outputTemplate": "[{Timestamp:HH:mm:ss} {Level:u3}] {Message:lj} {NewLine}{Exception}"

}

},

{

"Name": "File",

"Args": {

"path": "logs/log-.json",

"formatter": "Serilog.Formatting.Compact.CompactJsonFormatter, Serilog.Formatting.Compact",

"rollingInterval": "Day",

"retainedFileCountLimit": 7

}

}

],

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

}}

This implementation provides:

Rich contextual logging with class/method/line information

Performance tracking for method execution times

Structured logging with JSON format

Exception details enrichment

Flexible configuration through appsettings.json

Thread-safe logging operations

The solution is designed to work well in both development and production environments, with the ability to adjust log levels and outputs through configuration.