# 🛠 Step-by-Step: Ultra-Fast Enrichment with Caller Info

### 1. 📦 CallerInfoEnricher (NO StackTrace)

csharp

CopyEdit

using Serilog.Core;using Serilog.Events;using System.Collections.Concurrent;

public class CallerInfoEnricher : ILogEventEnricher

{

private static readonly ConcurrentDictionary<string, LogEventProperty> \_cache = new();

private readonly string \_memberName;

private readonly string \_filePath;

private readonly int \_lineNumber;

public CallerInfoEnricher(string memberName, string filePath, int lineNumber)

{

\_memberName = memberName;

\_filePath = filePath;

\_lineNumber = lineNumber;

}

public void Enrich(LogEvent logEvent, ILogEventPropertyFactory propertyFactory)

{

var className = System.IO.Path.GetFileNameWithoutExtension(\_filePath) ?? "UnknownClass";

logEvent.AddPropertyIfAbsent(GetOrCreateProperty(propertyFactory, "ClassName", className));

logEvent.AddPropertyIfAbsent(GetOrCreateProperty(propertyFactory, "MethodName", \_memberName));

logEvent.AddPropertyIfAbsent(GetOrCreateProperty(propertyFactory, "LineNumber", \_lineNumber));

}

private static LogEventProperty GetOrCreateProperty(ILogEventPropertyFactory factory, string name, object value)

{

var key = $"{name}:{value}";

if (\_cache.TryGetValue(key, out var cached))

return cached;

var property = factory.CreateProperty(name, value);

\_cache[key] = property;

return property;

}

}

**Notes:**

We cache properties for **extreme speed**.

No reflection, no stack traces = almost free runtime cost.

### 2. ⚡ LoggerExtensions (Extension Method to add Enricher easily)

csharp

CopyEdit

using System.Runtime.CompilerServices;using Serilog;

public static class LoggerExtensions

{

public static ILogger WithCallerInfo(this ILogger logger,

[CallerMemberName] string memberName = "",

[CallerFilePath] string filePath = "",

[CallerLineNumber] int lineNumber = 0)

{

return logger.ForContext(new CallerInfoEnricher(memberName, filePath, lineNumber));

}

}

Now you can **easily** call Log.WithCallerInfo().Information("message") 🚀

### 3. 🛠 PerformanceLoggingExtensions (Auto-inject Caller Info!)

csharp

CopyEdit

using System.Diagnostics;using Serilog;

public static class PerformanceLoggingExtensions

{

public static async Task<T> ExecuteWithPerformanceLoggingAsync<T>(

this Func<Task<T>> func,

string operationName,

ILogger logger = null,

[CallerMemberName] string memberName = "",

[CallerFilePath] string filePath = "",

[CallerLineNumber] int lineNumber = 0)

{

logger ??= Log.Logger;

var stopwatch = Stopwatch.StartNew();

try

{

T result = await func();

stopwatch.Stop();

logger.WithCallerInfo(memberName, filePath, lineNumber)

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

operationName, stopwatch.ElapsedMilliseconds);

return result;

}

catch (Exception ex)

{

stopwatch.Stop();

logger.WithCallerInfo(memberName, filePath, lineNumber)

.Error(ex, "Operation {OperationName} failed after {ElapsedMilliseconds} ms",

operationName, stopwatch.ElapsedMilliseconds);

throw;

}

}

public static async Task ExecuteWithPerformanceLoggingAsync(

this Func<Task> func,

string operationName,

ILogger logger = null,

[CallerMemberName] string memberName = "",

[CallerFilePath] string filePath = "",

[CallerLineNumber] int lineNumber = 0)

{

logger ??= Log.Logger;

var stopwatch = Stopwatch.StartNew();

try

{

await func();

stopwatch.Stop();

logger.WithCallerInfo(memberName, filePath, lineNumber)

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

operationName, stopwatch.ElapsedMilliseconds);

}

catch (Exception ex)

{

stopwatch.Stop();

logger.WithCallerInfo(memberName, filePath, lineNumber)

.Error(ex, "Operation {OperationName} failed after {ElapsedMilliseconds} ms",

operationName, stopwatch.ElapsedMilliseconds);

throw;

}

}

}

### 4. 🔥 Example Usage

csharp

CopyEdit

var builder = WebApplication.CreateBuilder(args);

// Configure Serilog

LoggingConfiguration.ConfigureLogging();

try

{

Log.Information("Starting up...");

var app = builder.Build();

app.MapGet("/", async () =>

{

await (() => Task.Delay(500))

.ExecuteWithPerformanceLoggingAsync("LoadingHomePage");

return "Hello World with Advanced Logging!";

});

app.Run();

}catch (Exception ex)

{

Log.Fatal(ex, "Application startup failed!");

}finally

{

Log.CloseAndFlush();

}

# 🏆 Resulting Log Example (Pretty!)

json

CopyEdit

{

"@t": "2025-04-28T12:34:56.789Z",

"@m": "Operation LoadingHomePage completed in 501 ms",

"ClassName": "Program",

"MethodName": "<MapGet>b\_\_0\_0",

"LineNumber": 25}

# 🚀 Summary of Differences from Previous Version

| **Traditional StackTrace** | **New Caller Info** |
| --- | --- |
| Heavy on CPU and memory | Super lightweight |
| Allocates stack frames | No allocations |
| Slower in high-throughput systems | Ultra fast |
| Might miss optimizations | Compiler-injected |

# ✅ Clean Architecture Diagram:

plaintext

CopyEdit

Program.cs

└── LoggingConfiguration.ConfigureLogging()

└── LoggerExtensions.WithCallerInfo()

└── CallerInfoEnricher (Inject ClassName, MethodName, LineNumber)

PerformanceLoggingExtensions (wrap method executions)

Would you also want me to show you how to **automatically log every method in a whole class** with AOP-style interception (without manually wrapping every method)? 🔥 (e.g., using DispatchProxy or lightweight Source Generators)?  
It’s super powerful for clean codebases! 🚀  
Should I show that too? 🎯