upGrad ENTERPRISE

Logging in .NET with C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. Introduction to Logging in .NET

What is Logging?

Logging is the process of recording application events for diagnostics, troubleshooting, auditing, or performance monitoring.

Why Use Logging?

- **Troubleshooting**: Track errors and bugs.
- **Auditing**: Record security-related events.
- **Monitoring**: Analyze system behavior and performance.

Basic Logging Example in C#

```
Console.WriteLine("Application started");
File.AppendAllText("log.txt", "Application started at " + DateTime.Now);
```

Problems with Manual Logging

- Hard to manage format and level
- Difficult to scale
- No filtering or querying

2. Structured Logging

What is Structured Logging?

Structured logging treats log entries as **data** (not just text). It captures context in a **queryable** format (like JSON).

Benefits

- Easily filter logs by fields (e.g., `UserId`, `OrderId`)
- Compatible with log aggregation tools (e.g., Seq, Kibana)
- Better analysis, visualization, and monitoring

Example Using Serilog:

```
Log.Information("User {UserId} placed order {OrderId}", 123, 456);
```

Log output (in JSON format):

```
{
  "Timestamp": "2025-07-24T10:00:00Z",
  "Level": "Information",
  "MessageTemplate": "User {UserId} placed order {OrderId}",
  "Properties": {
    "UserId": 123,
    "OrderId": 456
  }
}
```

3. Logging Frameworks in .NET

Common Logging Frameworks:

Framework	Features
Microsoft.Extensions.Logging	Built-in logging abstraction
Serilog	Structured logging, sinks (files, DB, console)
NLog	Powerful config support, targets
log4net	Apache project, traditional logging

Using Microsoft.Extensions.Logging (Console Example)

```
using Microsoft.Extensions.Logging;

class Program
{
    static void Main(string[] args)
    {
        using var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });

        ILogger logger = loggerFactory.CreateLogger<Program>();
        logger.LogInformation("App started");
    }
}
```

Using Serilog

```
using Serilog;

class Program
{
    static void Main(string[] args)
    {
        Log.Logger = new LoggerConfiguration()
            .WriteTo.Console()
            .WriteTo.File("log.txt")
            .CreateLogger();

        Log.Information("Starting application");
        Log.CloseAndFlush();
    }
}
```

4. Logging in Exception Handling

Logging Exceptions

You should always log exceptions with context (message, stack trace, severity).

Example:

```
try
{
    // some risky operation
    int result = 10 / int.Parse("0");
}
catch (Exception ex)
{
    Log.Error(ex, "An error occurred while performing division");
}
```

Logging Pipeline Architecture (Simplified)

```
[App Code]
    ↓
ILogger Interface
    ↓
Logging Provider (e.g., Serilog, Console)
    ↓
Sink (Console, File, DB, etc.)
```

Summary

Topic	Key Points
Logging Basics	Diagnostic and audit trail
Structured Logging	Logs as data, queryable
Logging Frameworks	Serilog, NLog, Microsoft.Extensions.Logging
Logging + Exceptions	Log all errors with context, avoid sensitive data

Working with Serilog in Console Application

Steps to implement Serilog in Console Application

Step 1: Create the Console App

Use either Visual Studio or the .NET CLI:

```
dotnet new console -n SerilogDemoApp
cd SerilogDemoApp
```

Step 2: Add Serilog NuGet Packages

Use the CLI:

```
dotnet add package Serilog
dotnet add package Serilog.Sinks.Console
dotnet add package Serilog.Sinks.File
```

Or use Visual Studio's NuGet Package Manager to install:

- `Serilog`
- `Serilog.Sinks.Console`
- `Serilog.Sinks.File`

Step 3: Program.cs Code

Replace `Program.cs` with the following:

```
using Serilog;
using System;

namespace SerilogDemoApp
{
    class Program
    {
        static void Main(string[] args)
        {
            // Configure Serilog
            Log.Logger = new LoggerConfiguration()
                .MinimumLevel.Debug()
                .WriteTo.Console()
                .WriteTo.File("logs/log.txt", rollingInterval: RollingInterval.Day)
                .CreateLogger();

            try
            {
                Log.Information("Application Starting Up");

                Console.WriteLine("Enter a number:");
                int number = int.Parse(Console.ReadLine());

                int result = 100 / number;

                Log.Information("Result of division is {Result}", result);
            }
            catch (Exception ex)
            {
                Log.Error(ex, "An error occurred while dividing");
            }
            finally
            {
                Log.Information("Application Ending");
                Log.CloseAndFlush();
            }
        }
    }
```

Output

Console:

```
[10:00:00 INF] Application Starting Up
[10:00:05 ERR] An error occurred while dividing
System.DivideByZeroException: Attempted to divide by zero.
...
```

File: `logs/log.txt`

Structured logs will also be written to file with timestamps and structured info.

Test Case

Run the app and:

- Input `0` → see exception logged.
- Input `10` → see correct result and structured logs.

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com