Controlling What Messages are Logged in ASP.NET Core Applications

Erik Dahl
PRINCIPAL ARCHITECT

@dahlsailrunner knowyourtoolset.com

Overview

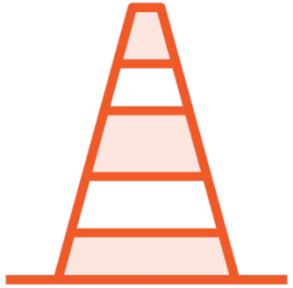
- Levels
- Categories and EventIds
- Filters
- Scopes
- High-Performance Situations

Log Levels

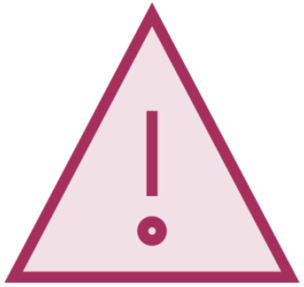
Trace / Verbose

Debug

Information



Warning



Error

Critical / Fatal

```
Log(LogLevel.Information, “logthis {stuff}”, stuff);

LogInformation(“logthis {stuff}”, stuff);
```

Logger Methods and Overloads

- **EventId: struct – Id, Name**
- **Exception**
- **Message: formatted string – allows {item})**
- **Args: Object[] - replacements for format items**

Categories and EventIds: Logical Grouping

Category

`ILogger<ClassName>`

Logged as "SourceContext"

Use `LoggerFactory` to define custom

Easy to see everything in "Category"

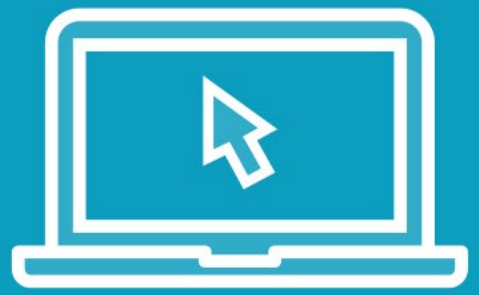
EventId

Officially struct with `Id` and `Name`

Can just be `Id` (`int`) – need lookup

Easy to see all of certain type of event

Demo



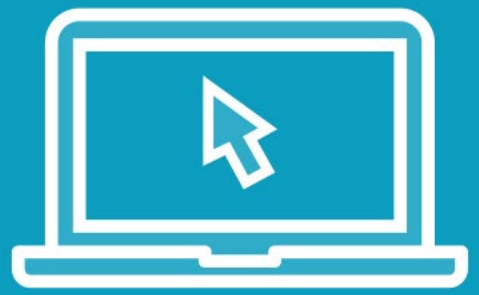
Updated application

- Google API to get book details
- Allow adding new books

More log entries!

- Different levels
- Explore categories and EventIds
- Utilize overload parameters

Demo



Exploring Filters

Syntax varies – concepts same

Minimum levels

- Global
- Per category
- Per provider / sink
- Per provider / sink and category

Other options exist for complex scenarios

```
using (_logger.BeginScope("Starting operation for {UserId}", userId))
{
    . . .
}
```

Scopes

- "Groups a set of logical operations"
- Shared content included in each log entry
- Spans class / assembly boundaries
- Start wherever you like

When it needs to be *Fast...*

Use `LoggerMessage` methods

- `LoggerMessage.Define`
- `LoggerMessage.DefineScope`

Avoids "boxing"

Templates only parsed once

http://bit.ly/high-perf-log-aspnetcore

Summary

Deep dive into Microsoft.Extensions.Logging

- Levels
- Categories and EventIds
- Filters
- Scopes
- LoggerMessage methods

Get DRY: Attributes, filters, and global handling