How and why to debug .NET applications - a subjective review of the possibilities

Beata Zalewa

zalnet

/bzalewa

ABOUT

In professional life consultant, Microsoft Certified Trainer, administrator, developer, freelancer, sometimes a miracle-worker and magician.

From the beginning of career associated with Microsoft technologies.

Private life in numbers: 1 husband, 1 daughter, 1 cat and 2 dogs. My hobbies are detective stories and photography.

Email: info@zalnet.pl

Skype: beata.zalewa

WWW: https://blog.zalnet.pl

zalnet

/bzalewa

AGENDA

- ✓ NLog
- ✓ Log4net
- ✓ Serilog
- ✓ Elmah
- ✓ BenchmarkDotNet package
- ✓ Working with LINQ
- ✓ MemoryDiagnoser
- ✓ Roslyn-based heap allocation analyzer

zalnet

/bzalewa

MOTTO

Filipe Fortes
@fortes

Follow

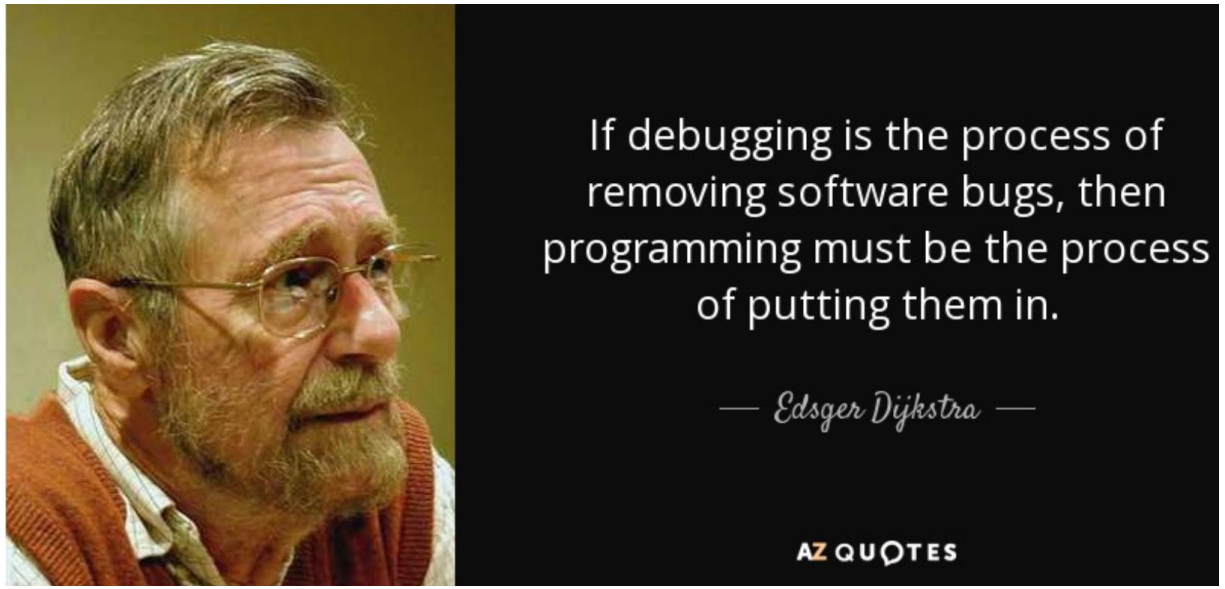
Debugging is like being the detective in a crime movie where you are also the murderer.

6:57 PM - 9 Nov 2013

zalnet

SOMETIMES THINGS GO WRONG IN PRODUCTION

- Event Logs
- Performance Monitor
- Reproduce (Testing & Staging)
- **Capture errors and analyze**
- **Logging** is one of the most basic things that every application needs to have to help troubleshoot application problems.



WHY SHOULD YOU USE C# LOGGING FRAMEWORK ?

- Most .NET developers know that they can write events to the Windows Event Viewer without implementing any extra libraries.
- This is the ultimate convenience and reliability.
- However, writing to the Event Viewer doesn't always work for some applications – for instance, you might want to write events to a log file to share with other departments or run analytics against a group of logs.
- Several logging frameworks promise .NET coders a "five-minute setup and start logging" solution for their projects.
- **Logging frameworks** are important because they make it easy to **send your logs to different places** by simply changing your configuration. You can write your logs to a text file on disk, a database, a log management system or potentially dozens of other places, all without changing your code.

LOGGING FRAMEWORK

- A logging framework is a utility specifically designed to standardize the process of logging in your application.
- This can come in the form of a third party tool, such as **log4j** or its .NET cousin, **log4net**.
- But a lot of organizations also roll their own.

LOGGING FRAMEWORK

- You have to go beyond just standardizing, however, to qualify as a framework.
- After all, you could simply "standardize" that logging meant invoking the file system API to write to a file called log.txt.
- A logging framework has to standardize the solution by taking it care of the logging for you, exposing a standard API.

/bzalewa

POSSIBLE LOGGING FRAMEWORKS

- The list of .NET Logging Framework: https://github.com/quozd/awesome-dotnet/blob/master/README.md#logging

- Essential Diagnostics - Extends the inbuilt features of System.Diagnostics namespace to provide flexible logging
- NLog - NLog - Advanced .NET and Silverlight logging
- Logazmic - Open source NLog viewer for Windows
- ELMAH - Official ELMAH site
- Elmah MVC - Elmah for MVC
- Logary - Logary is a high performance, multi-target logging, metric, tracing and health-check library for Mono and .NET. .NET's answer to DropWizard. Supports many targets, built for micro-services.
- Log4Net - The Apache log4net library is a tool to help the programmer output log statements to a variety of output targets
- Sentry - .NET SDK for Sentry Open-source error tracking that helps developers monitor and fix crashes in real time..
- Serilog - A no-nonsense logging library for the NoSQL era. Combines the best of traditional and structured diagnostic logging in an easy-to-use package.

zalnet

/bzalewa

TOP .NET LOGGING FRAMEWORKS ON THE MARKET

- NLog
- Log4net
- Microsoft Enterprise Library
- ELMAH
- NSpring
- Serilog
- log4net-loggly

zalnet

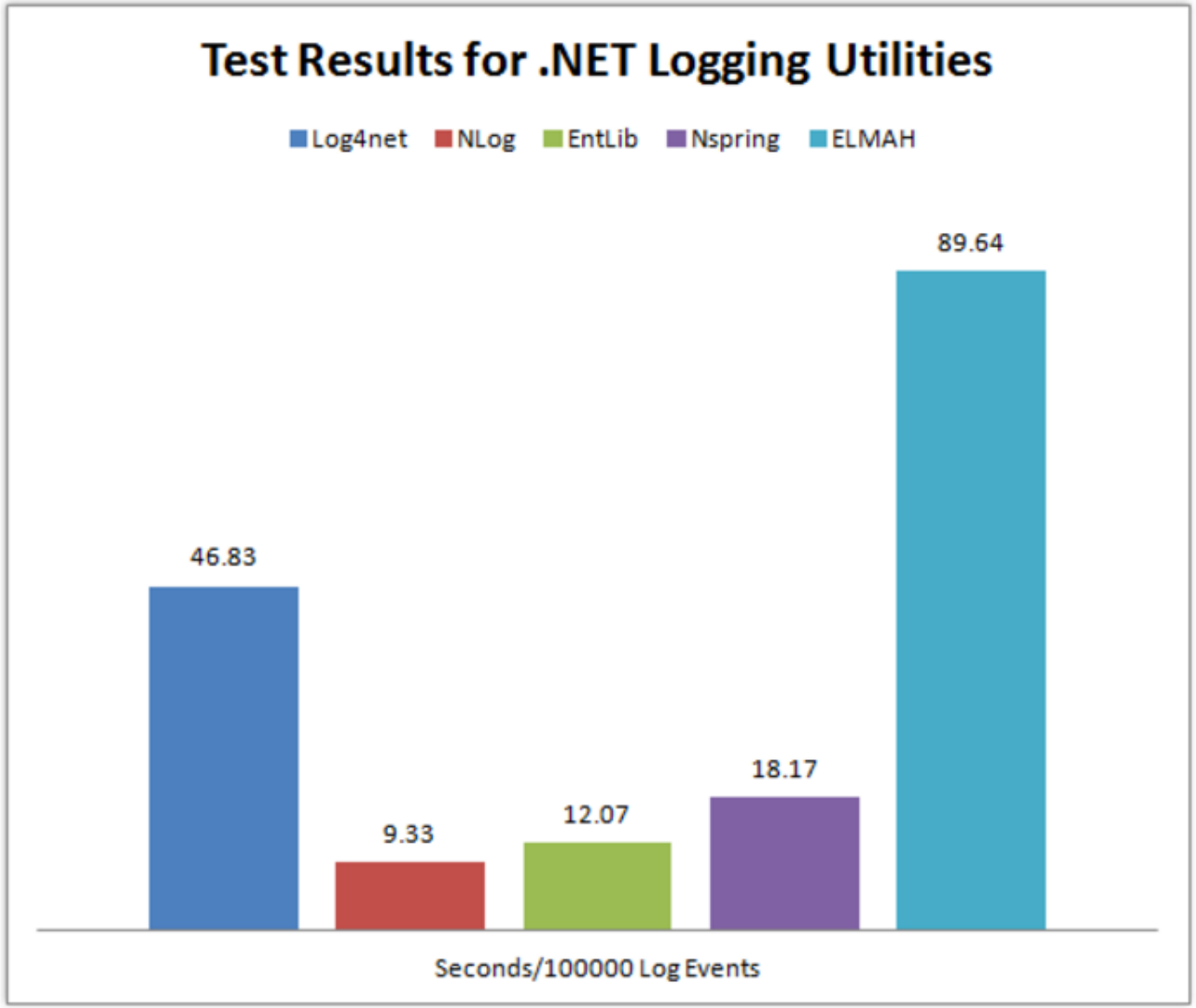
TOP .NET LOGGING FRAMEWORKS ON THE MARKET

- Log4net, NLog, Microsoft Enterprise Library, and NSpring can be used in console applications.
- ELMAH is a web logger, so this application was run against a web application in .NET 4.5.
- C# is the primary language framework.
- All libraries are available in NuGet, so developers didn't need to download specific libraries and integrate DLLs and third-party solutions into the main project.

/bzalewa

BENCHMARKS

(www.loggly.com)



zalnet

BENCHMARKS

- **NLog** was the fastest. NLog truly did stand by its claim that you could start logging events in minutes. Getting NLog to work with the application took two lines of code: one line to instantiate the main logger class, and the other to log the events.
- **Log4net** was just as convenient, but it lost in terms of performance. It was just as convenient as NLog, but it couldn't compete on performance.
- **NSpring** was easy to set up, but it required much more code to work with it. You must open and close NSpring logger events, which you don't need to do with the others. A few more lines of code in a simple application aren't too much of a hassle, but opening and closing logger variables can be tedious for enterprise applications.
- The native **Microsoft Enterprise Library** has the second fastest times, but it was extremely inconvenient. Microsoft recently changed the way its framework works from version 5 to 6. In version 5.0, the developer could make a static call to log functions, but now there are numerous obstacles to get EntLib to work with version 6.0. The setup time for EntLib downgrades this library even though it was the second fastest.
- The final framework is **ELMAH**. ELMAH is a web logger, so it functions a bit differently than the others. It writes to an XML file by default, which could also explain some of its lag.

CONCLUSION AFTER BENCHMARKS

- For heavy-hitting applications that require file logging and speed, NLog was clearly the winner.
- NLog also has good support from the community with integrations for log management solutions like Loggly.
- If for any reason NLog isn't your choice, go with NSpring instead.
- These two frameworks are clearly the fastest.
- We would have chosen log4net as our top choice had it not been for the slow performance. It was one of the easiest to set up.
- All of these frameworks are available in NuGet, so every developer can try them out for himself.

NLog

/bzalewa

NLOG

- NLog is a very popular logging framework for .NET. It is second only to Log4net in popularity but is much newer and has a few unique features. In opposite to Log4net, NLog has a lot of great features and advantages.
- NLog is a free logging platform for .NET with rich log routing and management capabilities. It makes it easy to produce and manage high-quality logs for your application regardless of its size or complexity.
- It can process diagnostic messages emitted from any .NET language, augment them with contextual information, format them according to your preference and send them to one or more targets such as file or database.
- .NET, C/C++ and COM interop APIs are supported so that all your application components including legacy modules written in C++/COM can route their log messages through a common engine.
- NLog is open source software, licensed under the terms of BSD license.
- Project Homepage: nlog-project.org
- Project RSS Feed: nlog-project.org/feed

zalnet

NLOG PACKAGES

NLog has several extension packages that add additional features.

The following packages are available:

- **NLog.Web** – package contains targets and layout-renders specific to ASP.NET 4 and IIS.
- **NLog.Web.AspNetCore** – package contains targets and layout-renders specific to ASP.NET Core and IIS.
- **NLog.Extensions.Logging** – .NET Core support.
- **NLog.Windows.Forms** – package contains targets specific for Windows.Forms.
- **NLog.Extended** – MSMQ target, AppSetting layout renderer.
- **NLog.Config** – NLog Configuration example/starter – not supported in .NET Core.
- **NLog.Schema** – XSD Schema for the NLog config xml – for, among other, Intellisense in Visual Studio – not supported in .NET Core.

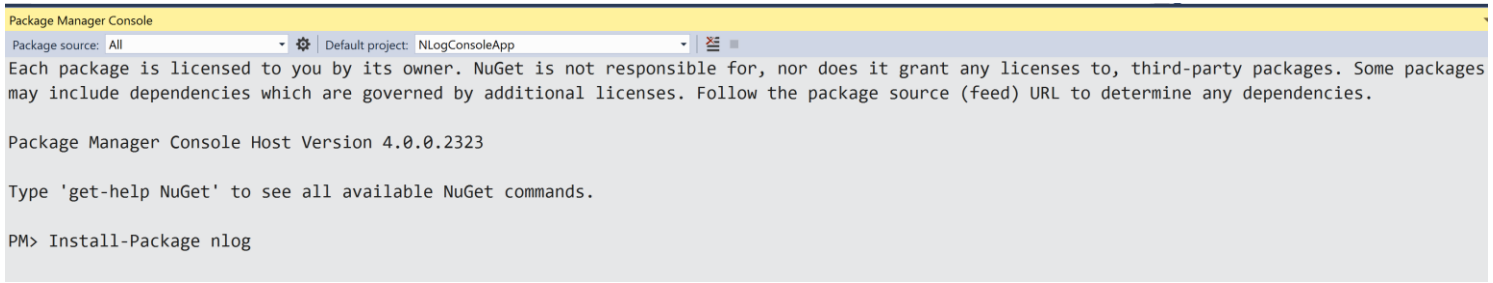
/bzalewa

NLOG INSTALLATION

You can download and install NLog via NuGet

- Starting with NLog is as easy as installing an NLog NuGet package.
- You will also want to pick some logging targets to direct where your log messages should go, including the console and a text file.
- Just install **NLog.Config** package and this will install also **NLog** and **NLog.Schema** packages - this will result in a starter config and Intellisense.

- Use the GUI or the following command in the Package Manager Console:



- That's it, you can now compile and run your application and it will be able to use NLog

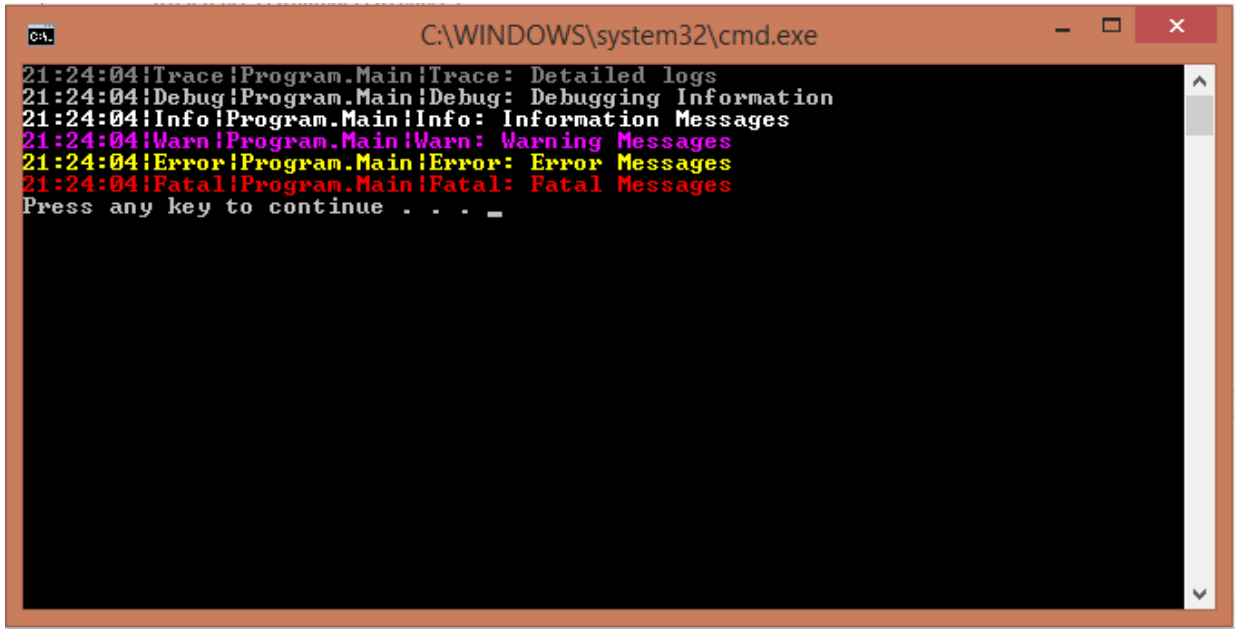
zalnet

/bzalewa

NLOG TARGETS

Logging Targets: What They Are and Common Targets You Need to Know

- Targets are how you direct where you want your logs sent.
- The most popular of the standard targets are most likely the File and Console targets.
- If you are using a Console, check out the ColoredConsole target:



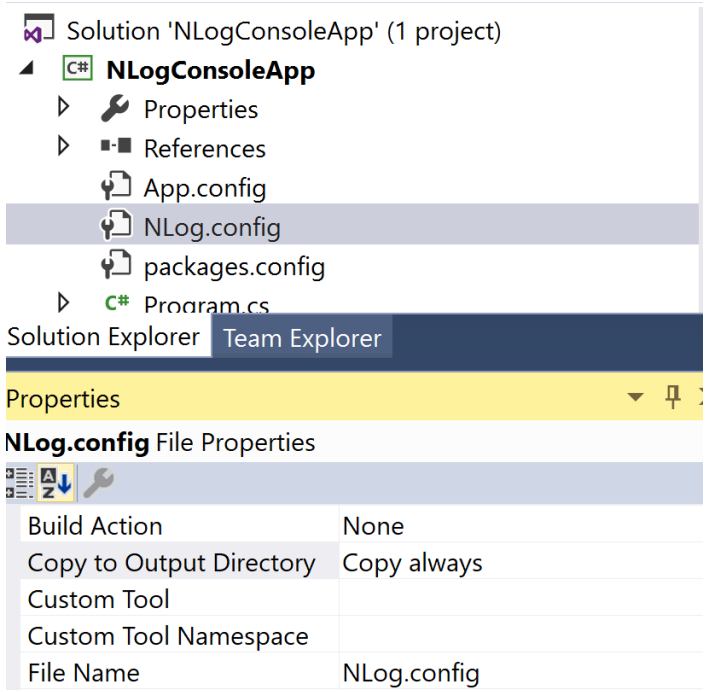
zalnet

/bzalewa

NLOG TARGETS

Configure NLog Targets for output

- NLog will only produce output if having configured one (or more) NLog targets.
- NLog can be configured using XML by adding a NLog.config file to your application project (File Properties: **Copy Always**).



zalnet

/bzalewa

NLOG TARGETS

Available targets:

Config options for NLog's configuration read more...

search all 187 items | All platforms

Targets 82 | Layouts 5 | Layout renderers 100

- ColoredConsole - Writes log messages to the console with customizable coloring.
- Console - Writes log messages to the console.
- File - Writes log messages to one or more files.
- Mail - Sends log messages by email using SMTP protocol or pickup folder.
- Mail - Mail Target using MailKit. Only SMTP supported. NLog.MailKit

Async and buffering

- AspNetBufferingWrapper - Buffers log events for the duration of ASP.NET request and sends them down to the wrapped target at the end of a request. NLog.Web
- AsyncWrapper - Provides asynchronous, buffered execution of target writes.
- AutoFlushWrapper - Causes a flush after each write on a wrapped target.
- BufferingWrapper - A target that buffers log events and sends them in batches to the wrapped target. Useful in combination with Mail target.

zalnet

/bzalewa

NLOG.CONFIG

This is a simple example of the content for NLog.config:

```
<?xml version="1.0" encoding="utf-8" ?>
  <nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <targets>
      <target name="logfile" xsi:type="File" fileName="file.txt" />
      <target name="logconsole" xsi:type="Console" />
    </targets>

    <rules>
      <logger name="*" minlevel="Info" writeTo="logconsole" />
      <logger name="*" minlevel="Debug" writeTo="logfile" />
    </rules>
  </nlog>
```

zalnet

/bzalewa

NLOG MESSAGES

Writing log messages

- Example of how to acquire a logger and writing a message to the logger:

```
var logger = NLog.LogManager.GetCurrentClassLogger();
logger.Info("Hello World");
```

- If having configured the NLog targets correctly, then it will output the message to the configured targets.

- The logger has a name, which is used by the logging-rules (**name="*"** is in configuration example above) to redirect to the wanted targets.

zalnet

/bzalewa

NLOG MESSAGES

- When using **NLog.LogManager.GetCurrentClassLogger(),** then it will create a logger with the name of the calling class (with namespace). One can also specify an explicit name by using NLog.LogManager.GetLogger("MyLogger").

- The logger can write messages with different LogLevels, which is used by the logging-rules (**minLevel** in configuration example above) so only relevant messages are redirected to the wanted targets.

- The logger is not tied to a specific target. The messages written to one logger can reach multiple targets based on the logging-rules configuration.

- Maintaining this separation lets you keep logging statements in your code and easily change how and where the logs are written, just by updating the configuration in one place.

zalnet

/bzalewa

NLOG LEVELS

Log levels

- Each log message has associated log level, which identifies how important/detailed the message is. NLog can route log messages based primarily on their logger name and log level.
- NLog supports the following log levels:

Trace - very detailed logs, which may include high-volume information such as protocol payloads. This log level is typically only enabled during development.

Debug - debugging information, less detailed than trace, typically not enabled in production environment.

Info - information messages, which are normally enabled in production environment.

Warn - warning messages, typically for non-critical issues, which can be recovered or which are temporary failures.

Error - error messages - most of the time these are Exceptions.

Fatal - very serious errors!

zalnet

/bzalewa

NLOG LAYOUT

Layouts and LayoutRenderers

It is possible to configure how log messages are written to a NLog target.

This shows the default SimpleLayout used by most NLog targets:

```
<target name="logfile" xsi:type="File" fileName="file.txt"
layout="${longdate}|${level:uppercase=true}|${logger}|${messag
e}" />
```

This can be configured to include more details:

```
<target name="logfile" xsi:type="File" fileName="file.txt"
layout="${longdate}|${level:uppercase=true}|${logger}|${threadid
}|${message}|${exception:format=tostring}" />
```

zalnet

/bzalewa

NLOG LAYOUT

Full list of Layout renderers is available on:

https://nlog-project.org/config

Config options for NLog's configuration read more...

search all 187 items | All platforms

Targets 82 | Layouts 5 | Layout renderers 100

- ${cached} - Applies caching to another layout output.
- ${exception} - Exception information provided through a call to one of the Logger methods
- ${level} - The log level (e.g. ERROR, DEBUG) or level ordinal (number)
- ${literal} - A string literal. (text) - useful to escape brackets
- ${logger} - The logger name. GetLogger, GetCurrentClassLogger etc
- ${message} - The (formatted) log message.
- ${newline} - A newline literal.
- ${onexception} - Only outputs the inner layout when exception has been defined for log message.
- ${var} - Render variable

zalnet

/bzalewa

NLOG LAYOUT

One can also use a more complex Layout instead of SimpleLayout:

CsvLayout

JsonLayout

CompoundLayout

Log4JXmlLayout

SimpleLayout

Config options for NLog's configuration read more...

search all 187 items | All platforms

Targets 82 | Layouts 5 | Layout renderers 100

- CSV - A specialized layout that renders CSV-formatted events.
- Compound - A layout containing one or more nested layouts.
- JSON - A specialized layout that renders to JSON.
- Log4JXml - A specialized layout that renders Log4j-compatible XML events.
- Simple (plain text) - The default layout when no Layout type is specified.

zalnet

/bzalewa

NLOG WRAPPERS

- **Wrappers**
- NLog supports special kinds of targets which do not do any logging by themselves, but which modify the behaviour of other loggers. Those targets are called wrappers. The most commonly used ones are:
- **AsyncWrapper** - Improves Performance by providing asynchronous, buffered execution of target writes.
- **BufferingWrapper** - Simple batching of log messages. Maybe only send batch when certain log event occurs (Ex. Exception).
- **FallbackGroup** - Provides fallback-on-error.
- **RetryingWrapper** - Provides retry-on-error.

Async and buffering

- AspNetBufferingWrapper - Buffers log events for the duration of ASP.NET request and sends them down to the wrapped target at the end of a request. NLog.Web
- AsyncWrapper - Provides asynchronous, buffered execution of target writes.
- AutoFlushWrapper - Causes a flush after each write on a wrapped target.
- BufferingWrapper - A target that buffers log events and sends them in batches to the wrapped target. Useful in combination with Mail target.

Databases

- Database - Writes log messages to the database using an ADO.NET provider.
- LiteDB - Writes NLog messages to LiteDB NLog.LiteDB External
- MongoDB - Writes NLog messages to MongoDB NLog.Mongo External

Filtering, throttling and error handling

- FallbackGroup - Provides fallback-on-error.
- FilteringWrapper - Filters log entries based on a condition.
- LimitingWrapper - Limits number of log events sent to target.
- PostFilteringWrapper - Filters buffered log entries based on a set of conditions that are evaluated on a group of events.
- RandomizeGroup - Sends log messages to a randomly selected target.
- RepeatingWrapper - Repeats each log event the specified number of times.
- RetryingWrapper - Retries in case of write error.
- RoundRobinGroup - Distributes log events to targets in a round-robin fashion.
- SplitGroup - Writes log events to all targets.

zalnet

/bzalewa

NLOG

Enable NLog's Own Internal Debug Logging

- If you are having any problems, you should enable the internal logging to help figure out the root cause.

```
<nlog internalLogFile="c:log.txt" internalLogLevel="Trace">
   <targets>
      <!-- target configuration here -->
   </targets>
   <rules>
      <!-- log routing rules -->
   </rules>
</nlog>
```

zalnet

NLOG RESULTS

- **Results**

C:\Windows\system32\cmd.exe

```
2018-10-14 21:41:16.9723|INFO|NLogConsoleApp.Program|Hello World
```

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <targets>
    <target name="logfile" xsi:type="File" fileName="file.txt" />
    <target name="logconsole" xsi:type="Console" />
  </targets>

  <rules>
    <logger name="*" minlevel="Info" writeTo="logconsole" />
    <logger name="*" minlevel="Debug" writeTo="logfile" />
  </rules>
</nlog>
```

DEMO

NLogConsoleApp

Log4net

/bzalewa

LOG4NET

What is Apache log4net™

- The Apache log4net library is a tool to help the programmer output log statements to a variety of output targets.
- log4net is a port of the excellent Apache log4j™ framework to the Microsoft® .NET runtime.
- The framework is similar in spirit to the original log4j while taking advantage of new features in the .NET runtime.

- **The Apache log4net project**
- log4net is part of the **Apache Logging Services** project at the **Apache Software Foundation**.
- The Logging Services project is intended to provide cross-language logging services for purposes of application debugging and auditing.

- Project website: http://logging.apache.org/log4net/

zalnet

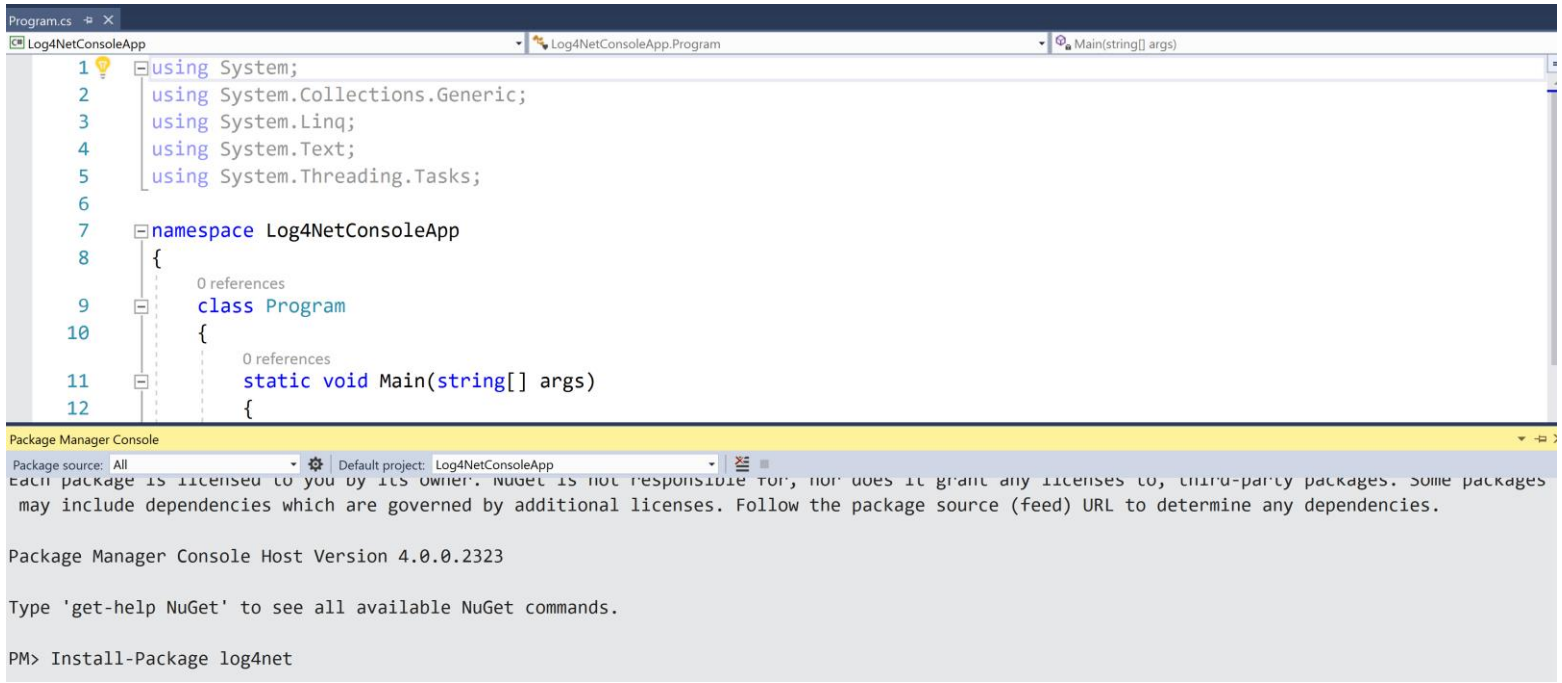
/bzalewa

HOW TO INSTALL LOG4NET VIA NUGET

1. Add log4net Package

- Starting with log4net is as easy as installing a NuGet package. You can use the Visual Studio UI to search for it and install it, or just run this quick command from the Package Manager Console.

PM> Install-Package log4net



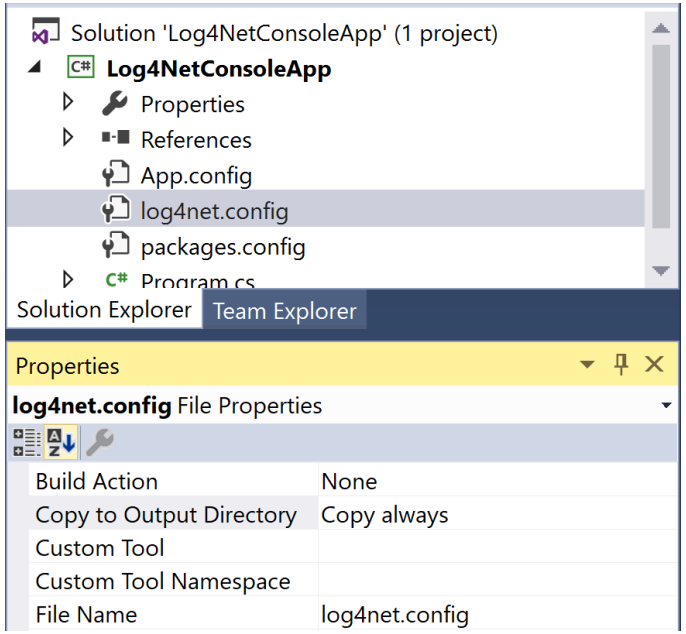
zalnet

/bzalewa

LOG4NET. CONFIG

2. Add log4net.config file

- Add a new file(*Application Configuration File*) to your project in Visual Studio called **log4net.config** and be sure to set a property for the file. Set "Copy to Output Directory" to "Copy Always". Then add content to this file.
- This is important because we need the **log4net.config** file to be copied to the bin folder when you build and run your app.



zalnet

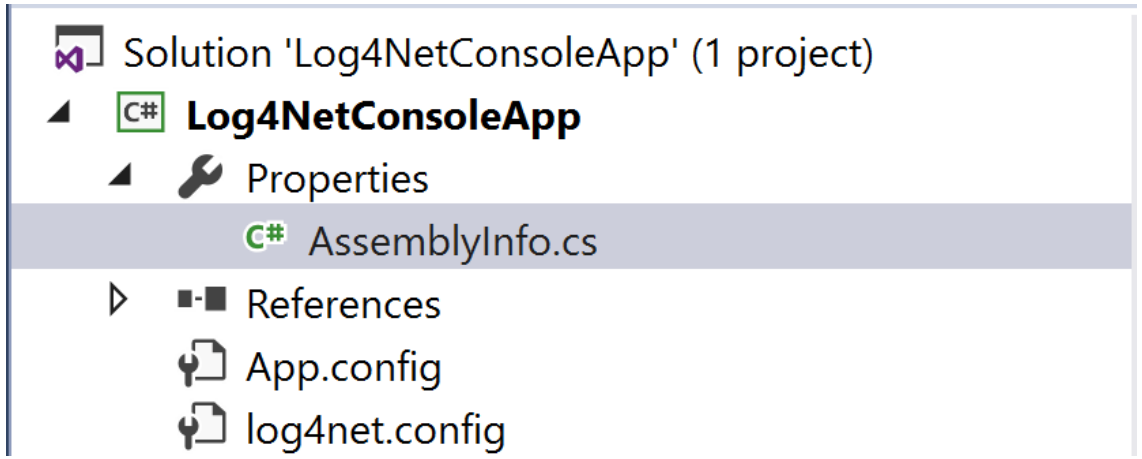
/bzalewa

LOG4NET

3. Tell log4net to Load Your Config

- The next thing we need to do is tell log4net where to load it's configuration from so that it actually works. I suggest putting this in your **AssemblyInfo.cs** file.
- Add this to the bottom of your AssemblyInfo file.

```
[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]
```



zalnet

/bzalewa

LOG4NET

4. Log Something!

- Now you can modify your app to log something and try it out!

```
using System;

namespace Log4NetConsoleApp
{
    0 references
    class Program
    {
        private static readonly log4net.ILog log = log4net.LogManager.GetLogger
          (System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);

        0 references
        static void Main(string[] args)
        {
            log.Info("Hello logging world!");
            Console.WriteLine("Please, click the Enter key");
            Console.ReadLine();
        }
    }
}
```

zalnet

/bzalewa

LOG4NET APPENDERS

Log Appenders: What They Are and Which Ones You Need to Know

- Appenders are how you direct where you want your logs sent.
- The most popular of the standard appenders are most likely the RollingFileAppender and ConsoleAppender.
- I would also try the DebugAppender if you want to see your log statements in the Visual Studio Debug window so you don't have to open a log file.
- If you are using a Console, checkout the ColoredConsoleAppender.

https://logging.apache.org/log4net/release/manual/introduction.html

Type	Description
log4net.Appender.AdoNetAppender	Writes logging events to a database using either prepared statements or stored procedures.
log4net.Appender.AnsiColorTerminalAppender	Writes color highlighted logging events to a an ANSI terminal window.
log4net.Appender.AspNetTraceAppender	Writes logging events to the ASP trace context. These can then be rendered at the end of the ASP page or on the ASP trace page.
log4net.Appender.BufferingForwardingAppender	Buffers logging events before forwarding them to child appenders.
log4net.Appender.ColoredConsoleAppender	Writes logging events to the application's Console. The events may go to either the standard our stream or the standard error stream. The events may have configurable text and background colors defined for each level.
log4net.Appender.ConsoleAppender	Writes logging events to the application's Console. The events may go to either the standard our stream or the standard error stream.
log4net.Appender.DebugAppender	Writes logging events to the .NET system.
log4net.Appender.EventLogAppender	Writes logging events to the Windows Event Log.
log4net.Appender.FileAppender	Writes logging events to a file in the file system.

/bzalewa

LOG4NET LEVELS

Log4net levels:

- **All** – Log everything
- **Debug**
- **Info**
- **Warn**
- **Error**
- **Fatal**
- **Off** – Don't log anything

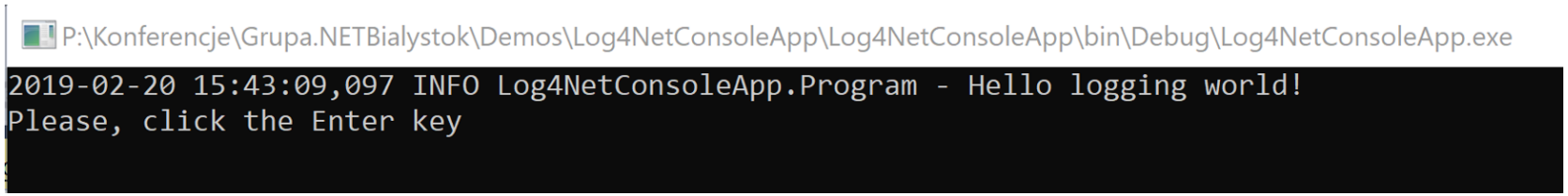
LOG4NET LEVELS

Make Good Use of Multiple Log Levels and Filter by Them

- Be sure to use Debug, Info, Warning, Error, and Fatal logging levels as appropriate within your code.
- Don't log everything as Debug.
- Be sure to think about how you would be viewing the logs and what you want to see it later when coding your logging statements.
- You can specify in your log4net config which log4net logging levels you want to log.
- This is really valuable if you want to specify only certain levels to be logged to a specific log appender or to reduce logging in production.
- This also you to log more or less data without changing your code.

LOG4NET

Result:



DEMO

Log4NetConsoleApp

/bzalewa

Serilog

zalnet

SERILOG

- The newest logging framework, 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.
- Serilog is a good logging framework and it is built with the structured log data in mind.
- It is a kind of serializer. Serilog determines the right representation when the properties are specified in log events.
- Serilog provides variety of sinks such as file, MSSQL, Log4net, PostgreSQL, etc.

SERILOG

- **Structured logging example**
- Often you'll find that you're writing logs that contain two things: a message and value.
- **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.
- **log.Debug("User id is {@userId}", userId);**

SERILOG

- Serilog is backed by a commercial company https://getseq.net/ .
- Retrace (log aggregator) also supports preserving the properties in the logs.
- Installing Serilog is slightly more complex than NLog or Log4net because it strives to be highly modular.

List of available sinks

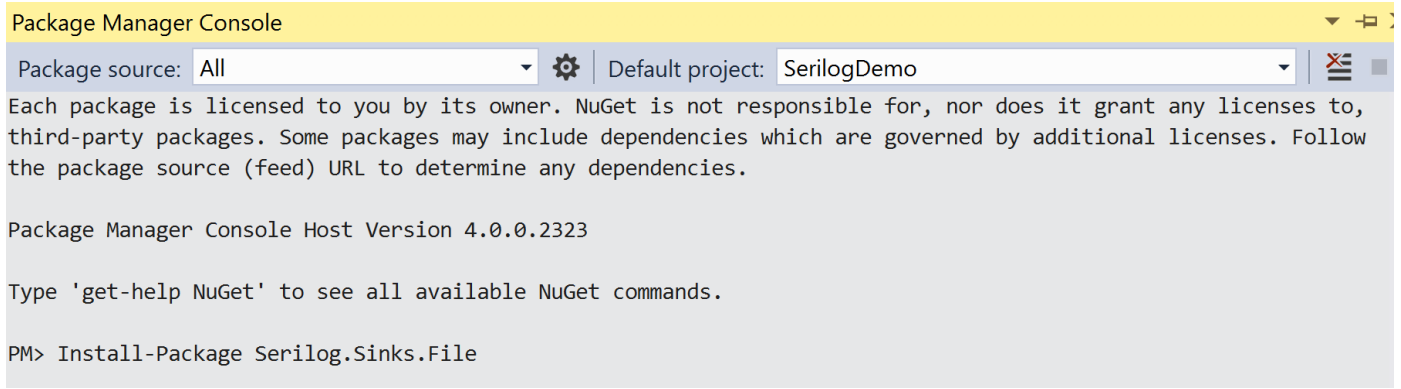
`*` Denotes sink is maintained by the wider Serilog community

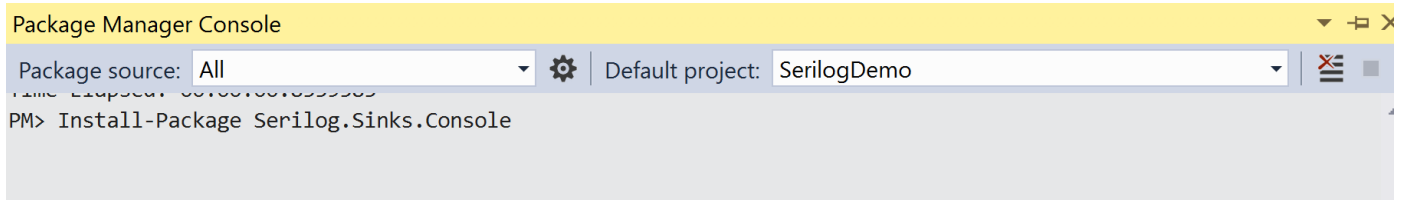
Sink Name	`WriteTo.*`	Package
Akka Actor *	`AkkaActor`	Serilog.Sinks.AkkaActor nuget v1.0.0.3 523
Alternate Rolling File *	`RollingFileAlternate`	Serilog.Sinks.RollingFileAlternate nuget v2.0.9 187.3k
Amazon CloudWatch *	`AmazonCloudWatch`	Serilog.Sinks.AwsCloudWatch nuget v4.0.138 230.5k
Amazon DynamoDB	`DynamoDB`	Serilog.Sinks.DynamoDB nuget v0.2.12 1.4k
Amazon Kinesis	`AmazonKinesis`	Serilog.Sinks.AmazonKinesis nuget v2.2.118 12.6k
Application Insights	`ApplicationInsights`	Serilog.Sinks.ApplicationinSights nuget v2.6.0 1.10m
Async Wrapper	`Async`	Serilog.Sinks.Async nuget v1.3.0 1.62m
Azure Analytics *	`AzureAnalytics`	Serilog.Sinks.AzureAnalytics nuget v3.1.0 54.9k
Azure DocumentDB	`AzureDocumentDB`	Serilog.Sinks.AzureDocumentDB nuget v4.0.0 66.5k

/bzalewa

SERILOG LOGGING

- To write log data into the file, we need to use "Serilog.Sinks.File" dependency. It writes Serilog event to one or more file based on configuration. We can add this sinks by using either NuGet package manager or .net CLI.
- Using Package Manager





zalnet

/bzalewa

SERILOG CONFIG

```
namespace SerilogDemo
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Log.Logger = new LoggerConfiguration()
                .MinimumLevel.Debug()
                .WriteTo.Console()
                .WriteTo.File("logfile.log", rollingInterval: RollingInterval.Day)
                .CreateLogger();

            Log.Debug("Starting up");
            Log.Debug("Shutting down");

            Console.ReadLine();
        }
    }
}
```

zalnet

SERILOG CONFIG

- By default, log file size limit is 1GB but it can be increased or removed using the fileSizeLimitBytes parameter of WriteTo.File method.

```
.WriteTo.File("Logs/Example.txt", fileSizeLimitBytes: null)
```

- We can also create log file per year, month, day, hour, or minute by specify rollingInterval parameter to WriteTo.File method. In following example, I have define rollingInterval to RollingInterval.Day so it will create new log file every day.

```
.WriteTo.File("Logs/Example.txt", rollingInterval: RollingInterval.Day)
```

- Serilog retained most recent 31 files by default for some reason. We can change or remove this limit using retainedFileCountLimit parameter. In following example, I have assign this parameter to null, so it remove limitation of 31 log files.

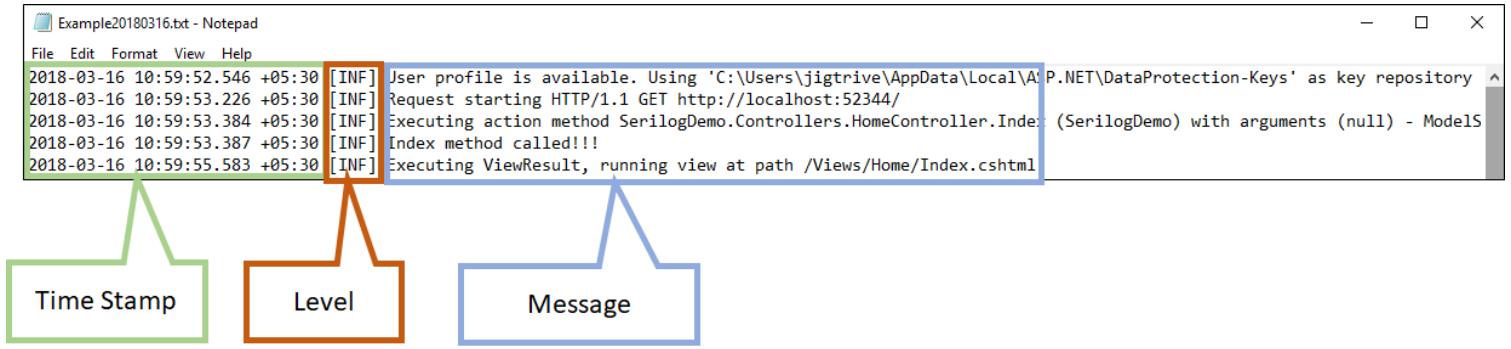
```
.WriteTo.File("Logs/Example.txt", rollingInterval: RollingInterval.Day, retainedFileCountLimit: null)
```

/bzalewa

SERILOG CONFIG

Controlling Text file formatting

- The File sink creates events in specific fix format by default. Following is fixed format .

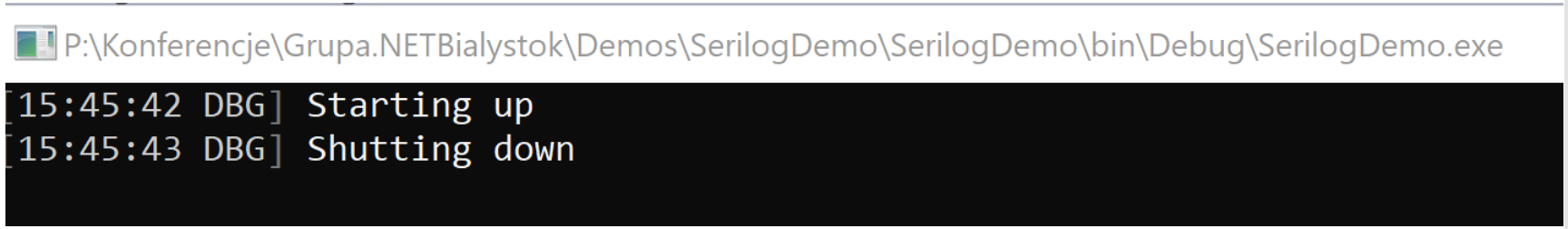


zalnet

/bzalewa

SERILOG LOGGING

- **Results:**



zalnet

DEMO

SerilogDemo

Elmah

/bzalewa

ELMAH INTRO

- **What is ELMAH ?**

  ELMAH stands for Error Logging Modules And Handlers that provide functionality to logging runtime ASP.NET errors.

- **Why do we choose ELMAH ?**

- ELMAH enables logging of all unhandled exceptions.

- ELMAH logs all errors in many storages, like - SQL Server, MySQL, Random Access Memory (RAM), SQL Lite, and Oracle.

- ELMAH has functionality to download all errors in CSV file.

- RSS feed for the last 15 errors

- Get all error data in JSON or XML format

- Get all errors to our mailbox

- Send error log notification to your application

- Customize the error log by customizing some code

zalnet

ELMAH INTRO

- **ELMAH HTTP Modules**

  There are three HTTP Modules:

- *ErrorMailModule*
  ErrorMailModule is used for sending the details of log as an email.

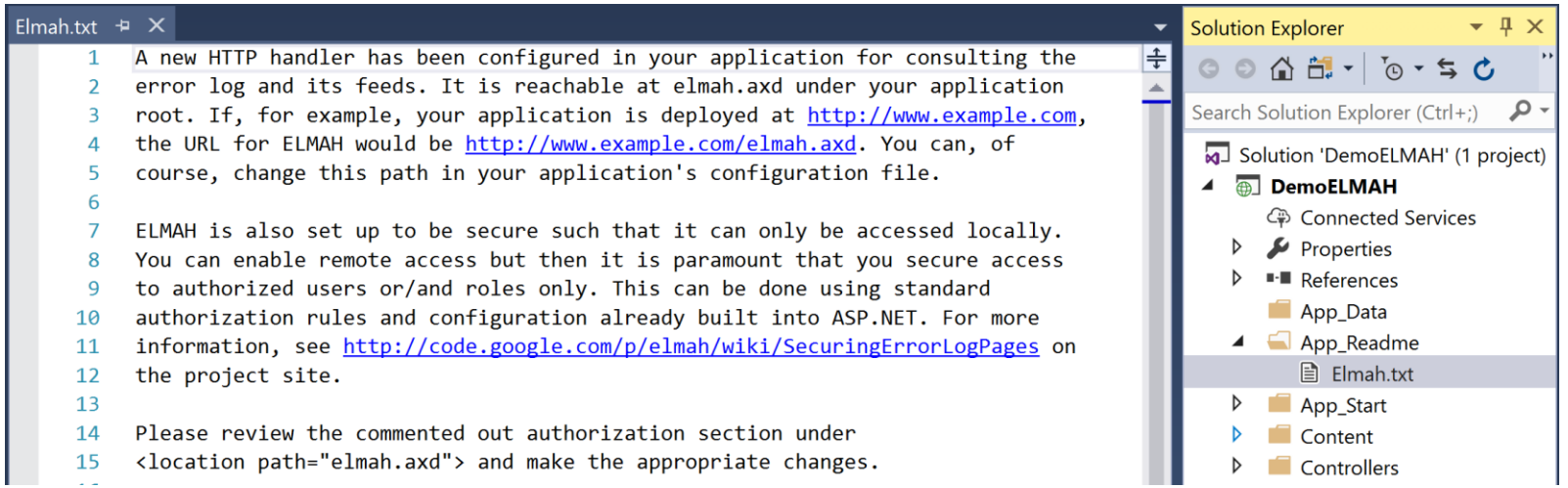
- *ErrorLogModule*
  ErrorLogModule is used for logging all the exceptions and some other details, like IP- Address, Username, Website Username etc.

- *ErrorFilterModule*
  ErrorFilterModule is used to customize the exception logs.

/bzalewa

ELMAH INSTALL

- After successful installation, we will find the below screen.



zalnet

/bzalewa

ELMAH CONFIG

- ELMAH Modules are default registered in web.config.

Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please
   visit
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <sectionGroup name="elmah">
      <section name="security" requirePermission="false"
       type="Elmah.SecuritySectionHandler, Elmah" />
      <section name="errorLog" requirePermission="false"
       type="Elmah.ErrorLogSectionHandler, Elmah" />
      <section name="errorMail" requirePermission="false"
       type="Elmah.ErrorMailSectionHandler, Elmah" />
      <section name="errorFilter" requirePermission="false"
       type="Elmah.ErrorFilterSectionHandler, Elmah" />
    </sectionGroup>
  </configSections>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2" />
    <httpModules>
      <add name="ErrorLog" type="Elmah.ErrorLogModule, Elmah" />
      <add name="ErrorMail" type="Elmah.ErrorMailModule, Elmah" />
      <add name="ErrorFilter" type="Elmah.ErrorFilterModule, Elmah" />
    </httpModules>
  </system.web>
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'DemoELMAH' (1 project)
  - DemoELMAH
    - Connected Services
    - Properties
    - References
    - App_Data
    - App_Readme
      - Elmah.txt
    - App_Start
    - Content
    - Controllers
    - ElmahLog
    - fonts
    - Models
    - Scripts
    - Views
    - favicon.ico
    - Global.asax
    - packages.config
    - Project_Readme.html
    - Web.config

zalnet

/bzalewa

ELMAH IN ACTION

- Create an **ActionMethod**.

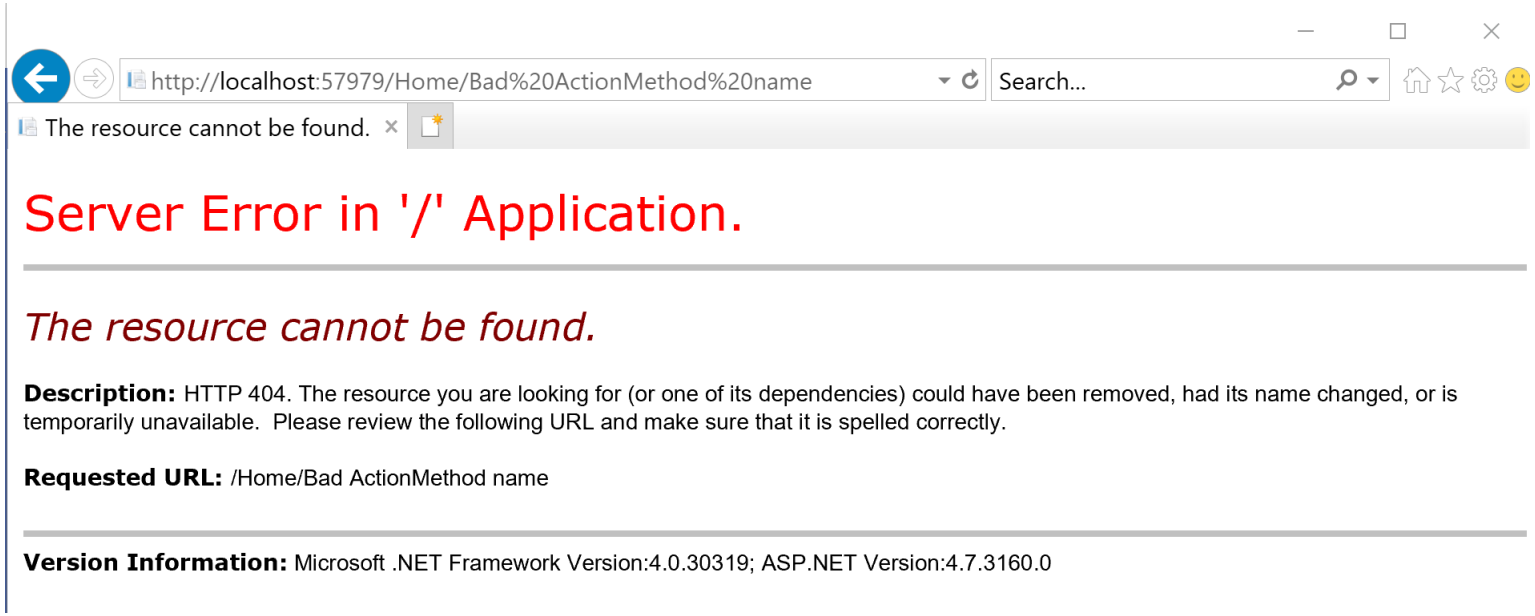
```
namespace DemoELMAH.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public ActionResult Index()
        {
            return View();
        }


    }
}
```

zalnet

/bzalewa

ELMAH IN ACTION

- Run the Project. Now, enter the wrong URL in address bar and hit Enter. We will get 404 Error.

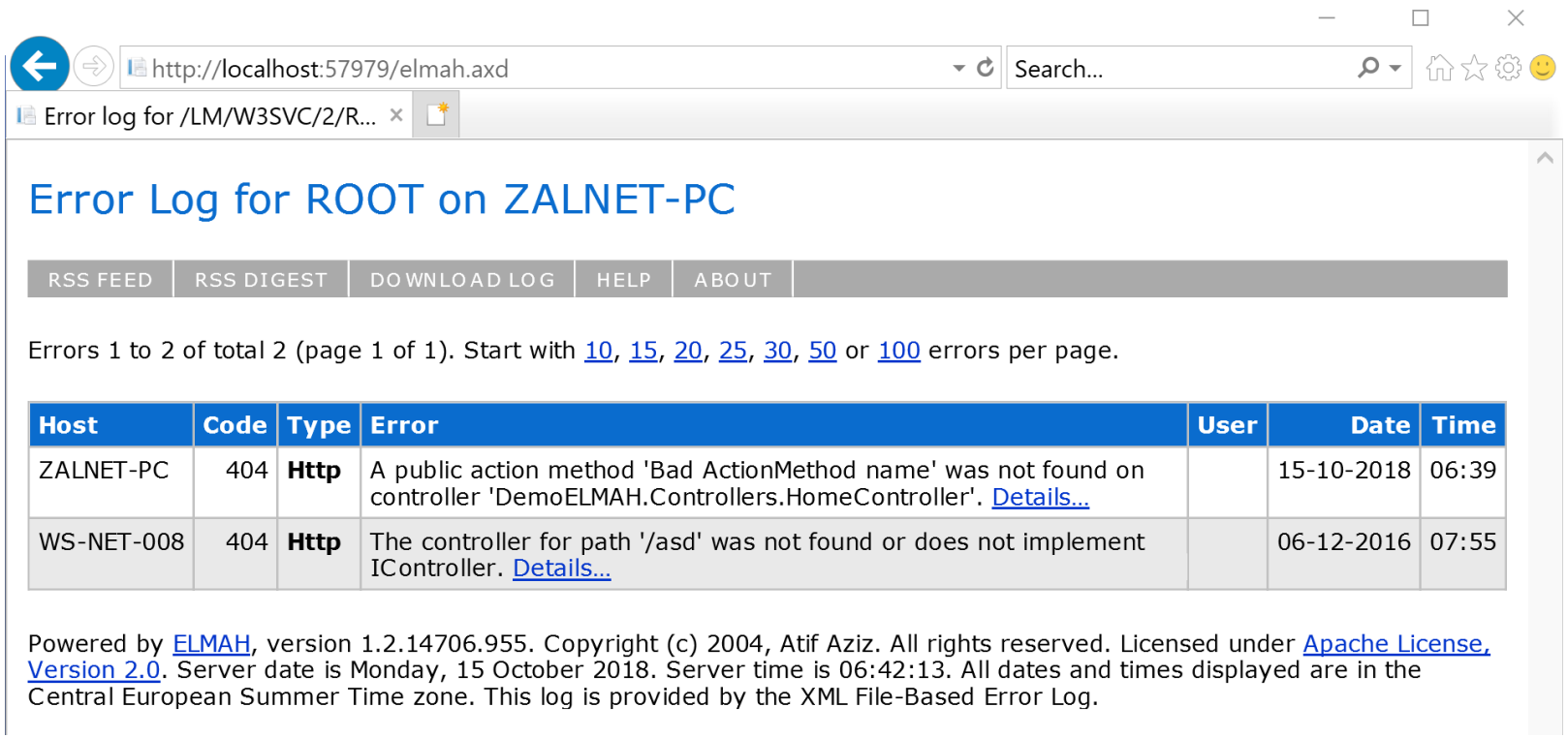


zalnet

/bzalewa

ELMAH IN ACTION

- Now, for ELMAH Log, enter the below URL in address bar.



Error Log for ROOT on ZALNET-PC

RSS FEED | RSS DIGEST | DOWNLOAD LOG | HELP | ABOUT

Errors 1 to 2 of total 2 (page 1 of 1). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

Host	Code	Type	Error	User	Date	Time
ZALNET-PC	404	**Http**	A public action method 'Bad ActionMethod name' was not found on controller 'DemoELMAH.Controllers.HomeController'. Details...		15-10-2018	06:39
WS-NET-008	404	**Http**	The controller for path '/asd' was not found or does not implement IController. Details...		06-12-2016	07:55

Powered by ELMAH, version 1.2.14706.955. Copyright (c) 2004, Atif Aziz. All rights reserved. Licensed under Apache License, Version 2.0. Server date is Monday, 15 October 2018. Server time is 06:42:13. All dates and times displayed are in the Central European Summer Time zone. This log is provided by the XML File-Based Error Log.

zalnet

/bzalewa

ELMAH MAIL

- Setup Mail Server for getting every log in email; add the below code in **web.config**.

```
<!--START: ELMAH Setting -->
<elmah>

  <errorLog type="Elmah.XmlFileErrorLog, Elmah" logPath="~/ElmahLog" />
  <errorMail from="info@zalnet.pl"
   to="info@zalnet.pl"
   subject="Error - ELMAH demo"
   async="true" />
  <!--
      See http://code.google.com/p/elmah/wiki/SecuringErrorLogPages for
      more information on remote access and securing ELMAH.
  -->
  <security allowRemoteAccess="false" />
</elmah>
<!--END: ELMAH Setting -->

<!--START: ELMAH Email Setting -->
<system.net>
  <mailSettings>
    <smtp deliveryMethod="Network">
      <network host="host address" port="12345" userName="username here" password="password here"/>
    </smtp>
  </mailSettings>
</system.net>
<!--END: ELMAH Email Setting -->
```

zalnet

/bzalewa

ELMAH LOCATIONS

- **Store ELMAH logs on different locations.**

Store ELMAH logs in XML file

- Create one folder named "ElmahLog" in root directory of your project. We use this folder for saving the XML file.

Add the below setting in your **Web.config**.

<elmah><errorLog type="Elmah.XmlFileErrorLog, Elmah" logPath="~/ElmahLog" /> </elmah>

Store ELMAH logs in RAM.

- Add the below setting in your **Web.config**.

<elmah><errorLog type="Elmah.MemoryErrorLog, Elmah" size="100" /> </elmah>

zalnet

/bzalewa

ELMAH LOCATIONS

Store ELMAH logs in Microsoft SQL Server.

- Add the below setting in your Web.config.

```
<elmah><errorLog type="Elmah.SqlErrorLog, Elmah"
connectionString="DBEntities" /> </elmah>
```

- Don’t forgot to add connectionString name as "DBEntities".

```
<connectionStrings><add name="DBEntities"
connectionString="data source=server name;initial
catalog=database name;persist security info=True;user id=your
username;password=your password;Trusted_Connection=True" />
</connectionStrings>
```

-

zalnet

/bzalewa

DEMO

DemoELMAH

zalnet

MONITORINF PERFORMANCE

BenchmarkDotNet package

COMMON PROGRAMMING CHALLENGE

- A common programming challenge is how to manage complexity around code performance – a small change might have a large impact on application performance.
- Sometimes you want to be able to perform micro-benchmarks against your code before and after and see really small changes, and know right away if you've made things better or worse.
- BenchmarkDotNet helps in this situation.

/bzalewa

HOW TO START?

Project with installed BenchmarkDotNet package and class witch Benchmark attribute:

```
using System;
using BenchmarkDotNet.Attributes;

namespace Services
{
    0 references
    public class RandomNumberGenerator : IRandomNumberGenerator
    {
        private static Random random = new Random();

        [Benchmark]
        2 references | 0 exceptions
        public int GetRandomNumber()
        {
            return random.Next();
        }

    }
}
```

zalnet

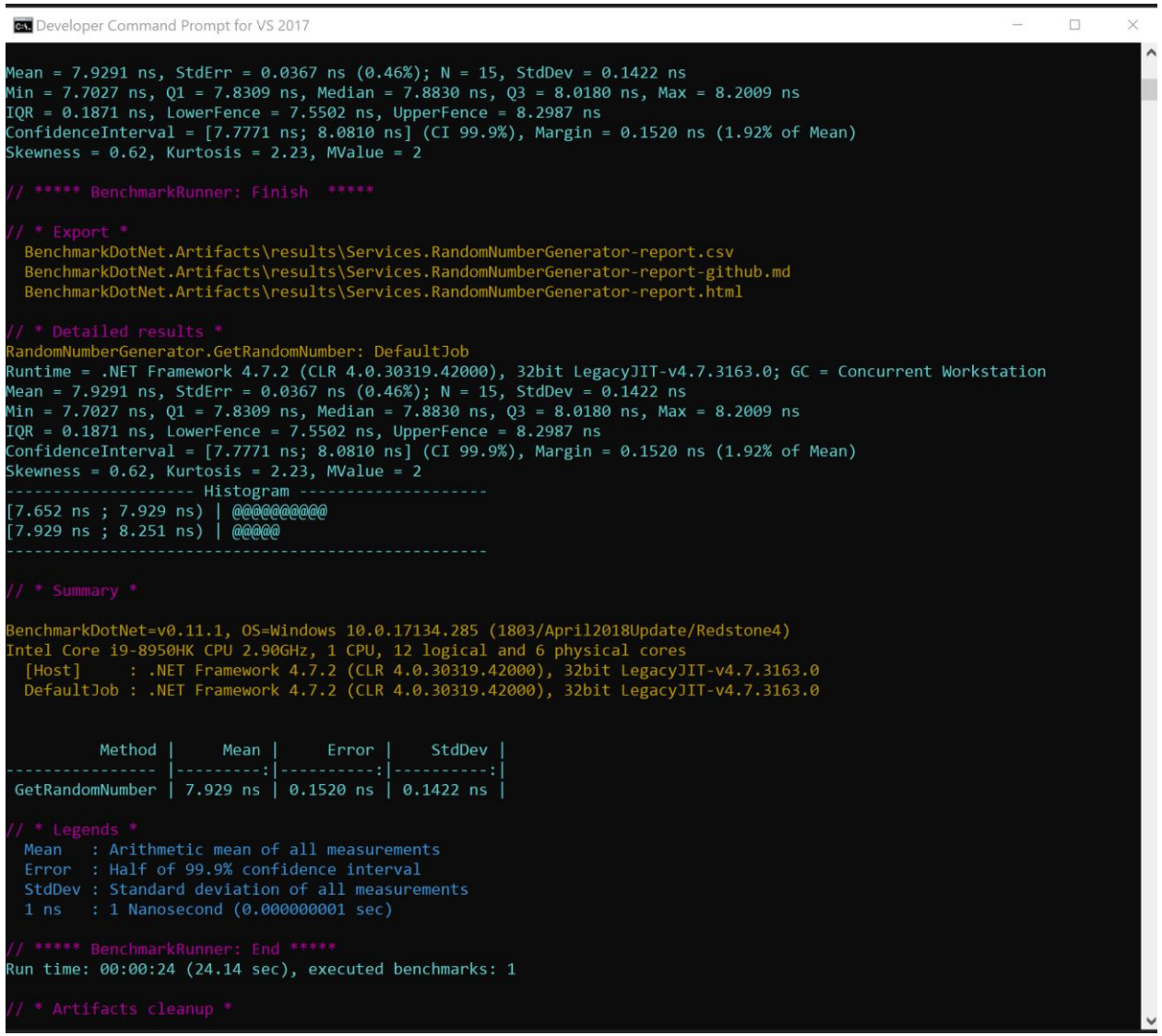
HOW TO START?

Call the BenchmarkRunner method:

```
using BenchmarkDotNet.Running;
using Services;


namespace PerformanceRunner
{
    0 references
    class Program
    {
        0 references | 0 exceptions
        static void Main(string[] args)
        {
            var summary = BenchmarkRunner.Run<RandomNumberGenerator>();
        }
    }
}
```

/bzalewa

RESULTS



zalnet

RESULTS

Developer Command Prompt for VS 2017

```
// * Summary *

BenchmarkDotNet=v0.11.1, OS=Windows 10.0.17134.285 (1803/April2018Update/Redstone4)
Intel Core i9-8950HK CPU 2.90GHz (Max: 1.50GHz), 1 CPU, 12 logical and 6 physical cores
  [Host]     : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0
  DefaultJob : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0


 Method | Number |      Mean |     Error |    StdDev |
------- |------- |----------:|----------:|----------:|
 Square |      1 | 0.0168 ns | 0.0217 ns | 0.0275 ns |
 Square |      2 | 0.0500 ns | 0.0386 ns | 0.0502 ns |

// * Hints *
Outliers
  MathFunctions.Square: Default -> 2 outliers were removed
  MathFunctions.Square: Default -> 1 outlier  was  removed

// * Legends *
  Number : Value of the 'Number' parameter
  Mean   : Arithmetic mean of all measurements
  Error  : Half of 99.9% confidence interval
  StdDev : Standard deviation of all measurements
  1 ns   : 1 Nanosecond (0.000000001 sec)

// ***** BenchmarkRunner: End *****
Run time: 00:01:58 (118.26 sec), executed benchmarks: 2

// * Artifacts cleanup *
```

DEMO

BenchmarkDemoA

BenchmarkDemoB

BenchmarkDemoC


/bzalewa
MONITORING
PERFORMANCE
Working with LINQ
zalnet

LINQ

- Language-Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language.
- Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support. Furthermore, you have to learn a different query language for each type of data source: SQL databases, XML documents, various Web services, and so on.
- With LINQ, a query is a first-class language construct, just like classes, methods, events.

/bzalewa

WHICH WAY IS FASTER?

Is a LINQ statement faster than a 'foreach' loop?

90

I am writing a Mesh Rendering manager and thought it would be a good idea to group all of the meshes which use the same shader and then render these while I'm in that shader pass.

I am currently using a `foreach` loop, but wondered if utilising LINQ might give me a performance increase?

15

c# performance linq foreach

share improve this question

edited Aug 29 '16 at 15:29
Stijn
15.3k ● 9 ● 74 ● 125

asked Jul 1 '10 at 8:18
Neil Knight
35.2k ● 19 ● 104 ● 175

1 possible duplicate of "Nested foreach" vs "lambda/linq query" performance(LINQ-to-Objects) – Daniel Earwicker Jul 1 '10 at 8:22

1 Please consider setting @MarcGravell's answer to the accepted one, there are situations, linq to sql for example, where linq is faster than the for/foreach. – paqogomez Oct 10 '14 at 15:50

add a comment

8 Answers

active oldest votes

163

Why should LINQ be faster? It also uses loops internally.

Most of the times, LINQ will be a bit slower because it introduces overhead. Do not use LINQ if you care much about performance. Use LINQ because you want shorter better readable and maintainable code.

zalnet

/bzalewa

HOW TO CHECK IT?

Compare 3 different queries with the same end result:

Administrator: Developer Command Prompt for VS 2017 - WorkingWithLINQ.exe

```
**********************************************************************
** Visual Studio 2017 Developer Command Prompt v15.0.26228.49
** Copyright (c) 2017 Microsoft Corporation
**********************************************************************

C:\Windows\System32>P:

P:\>cd P:\Konferencje\Grupa.NETBialystok\Demos\WorkingWithLINQ\WorkingWithLINQ\bin\Debug

P:\Konferencje\Grupa.NETBialystok\Demos\WorkingWithLINQ\WorkingWithLINQ\bin\Debug>WorkingWithLINQ.exe
3,54294
3,54294
3,54294
```

```
// * Summary *

BenchmarkDotNet=v0.11.1, OS=Windows 10.0.17134.285 (1803/April2018Update/Redstone4)
Intel Core i9-8950HK CPU 2.90GHz, 1 CPU, 12 logical and 6 physical cores
  [Host]     : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0
  DefaultJob : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0


                    Method |       Mean |    Error |   StdDev |
-------------------------- |-----------:|---------:|---------:|
          SumUsingForLoops |   69.42 ms | 1.385 ms | 3.154 ms |
 SumUsingForLoopsAndString |  105.06 ms | 1.213 ms | 1.134 ms |
             SumUsingLinq |  597.48 ms | 7.652 ms | 7.157 ms |
```

zalnet

/bzalewa

DEMO

Working with LINQ

zalnet

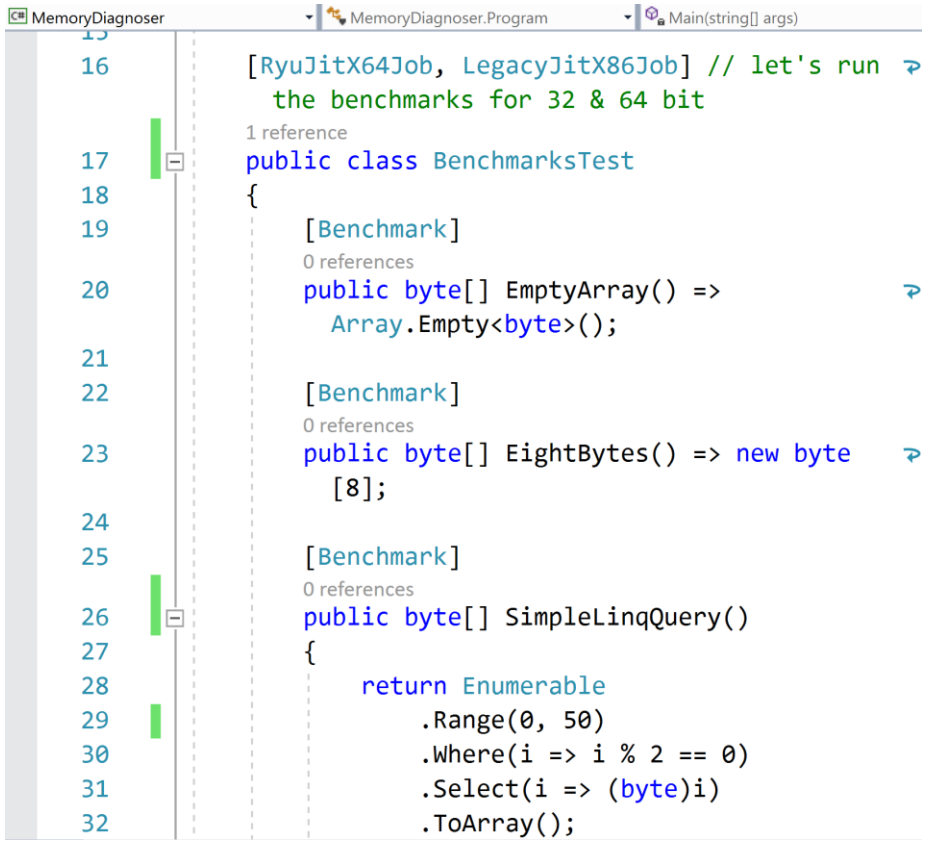
MemoryDiagnoser

MEMORY DIAGNOSER

- MemoryDiagnoser is one of its features that allows measuring the number of allocated bytes and garbage collection frequency.
- Before the **0.10.1** version of BenchmarkDotNet the MemoryDiagnoser was part of **BenchmarkDotNet.Diagnostics.Windows** package. Internally it was using Event Tracing for Windows (ETW), which had following implications:
- It was not cross-platform (**Windows only**).
- It was as accurate as ETW allowed us it to be. We were using the *GCAllocationTick* event which is raised *"Each time approximately 100 KB is allocated"*. Which means that if 199 KB was allocated we would know only about the first 100 KB.

MEMORY DIAGNOSER

- Class with Benchmark attribute:



MEMORY DIAGNOSER

Developer Command Prompt for VS 2017

```
// * Summary *

BenchmarkDotNet=v0.11.1, OS=Windows 10.0.17134.285 (1803/April2018Update/Redstone4)
Intel Core i9-8950HK CPU 2.90GHz, 1 CPU, 12 logical and 6 physical cores
  [Host]       : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0
  LegacyJitX86 : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 32bit LegacyJIT-v4.7.3163.0
  RyuJitX64    : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 64bit RyuJIT-v4.7.3163.0

Runtime=Clr

         Method |          Job |       Jit | Platform |        Mean |      Error |     StdDev |      Median |
--------------- |------------- |---------- |--------- |------------:|-----------:|-----------:|------------:|
     EmptyArray | LegacyJitX86 | LegacyJit |      X86 |   0.1744 ns |  0.0588 ns |  0.1340 ns |   0.1745 ns |
     EightBytes | LegacyJitX86 | LegacyJit |      X86 |   2.3960 ns |  0.1246 ns |  0.2962 ns |   2.2990 ns |
 SimpleLinqQuery | LegacyJitX86 | LegacyJit |      X86 | 755.0748 ns | 10.2032 ns |  9.5441 ns | 758.3945 ns |
     EmptyArray |    RyuJitX64 |    RyuJit |      X64 |   0.0031 ns |  0.0110 ns |  0.0113 ns |   0.0000 ns |
     EightBytes |    RyuJitX64 |    RyuJit |      X64 |   3.0174 ns |  0.1499 ns |  0.4372 ns |   2.9502 ns |
 SimpleLinqQuery |    RyuJitX64 |    RyuJit |      X64 | 669.3525 ns | 13.1610 ns | 25.6694 ns | 662.4184 ns |

// * Warnings *
MultimodalDistribution
  BenchmarksTest.EmptyArray: LegacyJitX86 -> It seems that the distribution is bimodal (mValue = 3.88)
  BenchmarksTest.EightBytes: RyuJitX64    -> It seems that the distribution is multimodal (mValue = 4.64)

// * Hints *
Outliers
  BenchmarksTest.EightBytes: LegacyJitX86   -> 2 outliers were removed
  BenchmarksTest.SimpleLinqQuery: RyuJitX64 -> 3 outliers were removed

// * Legends *
  Mean   : Arithmetic mean of all measurements
  Error  : Half of 99.9% confidence interval
  StdDev : Standard deviation of all measurements
  Median : Value separating the higher half of all measurements (50th percentile)
  1 ns   : 1 Nanosecond (0.000000001 sec)

// ***** BenchmarkRunner: End *****
Run time: 00:06:08 (368.12 sec), executed benchmarks: 6

// * Artifacts cleanup *

E:\Demo\MemoryDiagnoser\MemoryDiagnoser\bin\Release>
```

/bzalewa

MEMORY DIAGNOSER

- Add the MemoryDiagnoser attribute to the class :

```
MemoryDiagnoser | MemoryDiagnoser.BenchmarksTest | EmptyArray()
14      }
15      [MemoryDiagnoser]
16      [RyuJitX64Job, LegacyJitX86Job] // let's run
          the benchmarks for 32 & 64 bit
        1 reference
17      public class BenchmarksTest
18      {
19          [Benchmark]
            0 references
20          public byte[] EmptyArray() =>
              Array.Empty<byte>();
21
22          [Benchmark]
            0 references
23          public byte[] EightBytes() => new byte
              [8];
24
```

/bzalewa

MEMORY DIAGNOSER

Developer Command Prompt for VS 2017

```
Runtime=Clr

         Method |          Job |       Jit | Platform |        Mean |      Error |     StdDev |      Median |  Gen 0 | Allocated |
--------------- |------------- |---------- |--------- |------------:|-----------:|-----------:|------------:|-------:|----------:|
     EmptyArray | LegacyJitX86 | LegacyJit |      X86 |   0.1621 ns |  0.0603 ns |  0.1190 ns |   0.1564 ns |      - |       0 B |
     EightBytes | LegacyJitX86 | LegacyJit |      X86 |   2.4641 ns |  0.1067 ns |  0.0998 ns |   2.4346 ns | 0.0038 |      20 B |
 SimpleLinqQuery | LegacyJitX86 | LegacyJit |      X86 | 744.8013 ns | 16.2807 ns | 33.6224 ns | 729.5350 ns | 0.0486 |     256 B |
     EmptyArray |    RyuJitX64 |    RyuJit |      X64 |   0.0699 ns |  0.0442 ns |  0.0902 ns |   0.0235 ns |      - |       0 B |
     EightBytes |    RyuJitX64 |    RyuJit |      X64 |   2.9630 ns |  0.1274 ns |  0.1701 ns |   2.9737 ns | 0.0051 |      32 B |
 SimpleLinqQuery |    RyuJitX64 |    RyuJit |      X64 | 668.1365 ns | 13.5909 ns | 33.8460 ns | 670.4341 ns | 0.0601 |     384 B |


// * Warnings *
MultimodalDistribution
  BenchmarksTest.EmptyArray: LegacyJitX86 -> It seems that the distribution is bimodal (mValue = 4)
  BenchmarksTest.EightBytes: LegacyJitX86 -> It seems that the distribution can have several modes (mValue = 2.86)

// * Hints *
Outliers
  BenchmarksTest.SimpleLinqQuery: LegacyJitX86 -> 1 outlier  was  removed
  BenchmarksTest.EightBytes: RyuJitX64         -> 6 outliers were removed, 7 outliers were detected

// * Legends *
  Mean      : Arithmetic mean of all measurements
  Error     : Half of 99.9% confidence interval
  StdDev    : Standard deviation of all measurements
  Median    : Value separating the higher half of all measurements (50th percentile)
  Gen 0     : GC Generation 0 collects per 1k Operations
  Allocated : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 ns      : 1 Nanosecond (0.000000001 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
Run time: 00:05:34 (334.79 sec), executed benchmarks: 6
```

zalnet

Email: info@zalnet.pl www.zalnet.pl

DEMO

MemoryDiagnoser

Roslyn-based heap allocation analyzer

/bzalewa

zalnet

/bzalewa

CLR HEAP ALLOCATION ANALYZER

- Roslyn based C# heap allocation diagnostic analyzer.
- It can detect most heap allocations including explicit allocations, value type to reference type (boxing), closure captures (a.k.a Display Classes) and can tell you why the closure is being captured. Implicit delegate creation and implicit allocations done by the compiler for params, etc.
- It can also run as part of your build and flag as warnings. It is, however, most demonstrative in its code-assist form in the IDE.
- It can be downloaded from Visual Studio MarketPlace.
- https://github.com/mjsabby/RoslynClrHeapAllocationAnalyzer for the source.

zalnet

DEMO

HeapAnalyzer

/bzalewa

Q & A

Email: info@zalnet.pl

Demo:

https://github.com/bzalewa/90.SpotkanieBialostockiejGrupy.NET

Thank you for your precious time ☺

zalnet