## Comparison of logging frameworks

The following table shows how System.Diagnostics stacks up against some popular 3rd party logging frameworks, as well as the Enterprise Library Logging Application Block extensions from Microsoft, comparing the general features, the information logged, the filters that can be used and the output formats available.

The System.Diagnostics column indicates both the built in features, as well as the features available in the Essential.Diagnostics and other extensions.

Please contact me if you think any information in this table is out of date.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | System. Diagnostics | Log4net | NLog | Enterprise Library |
| **General Features** |  |  |  |  |
| Availability | built-in | 3rd party | 3rd party | Microsoft |
| Levels | 5 | 5 | 6 | 5 |
| Multiple sources | Yes | Yes | Yes | Yes |
| Hierarchical sources | - | Yes | Yes | - |
| Extensible | Yes | Yes | Yes | Yes |
| Listener chaining | - | Yes | Yes | Yes |
| Delayed formatting | Yes | Yes | Yes | - |
| Lambda | - | - | Yes | - |
| Templates | EX | Yes | Yes | Yes |
| Logging interface | EX | Yes | Yes | - |
| Dynamic configuration | EX | Yes | Yes | Yes |
| Minimum trust | - | TBA | TBA | - |
| Trace .NET framework (WCF, WIF, System.Net, etc) | Yes | - | - | - |
| Source from .NET Trace | n/a | - | - | Yes |
|  |  |  |  |  |
| **Log Information** |  |  |  |  |
| Event ID | Yes | contrib extension | - | Yes |
| Priority | - | - | - | Yes |
| Process/thread information | Yes | Yes | Yes | Yes |
| ASP.NET information | EX | Yes | Yes | Yes |
| Correlation identifier | Yes | via context | via context | Yes |
| Cross-process correlation | Yes | - | - | Yes |
| Exceptions | - | Yes | Yes | via exception block |
|  |  |  |  |  |
| **Filters** |  |  |  |  |
| Event level | Yes | Yes | Yes | Yes |
| Source | Yes | Yes | Yes | Yes |
| Property | EX | Yes | - | - |
| String match | - | Yes | Yes | - |
| Expression | EX | - | Yes | - |
| Priority | - | - | - | Yes |
|  |  |  |  |  |
| **Listeners** |  |  |  |  |
| ASP.NET Trace | Yes | Yes | Yes | - |
| Chainsaw (log4j) | - | Yes | Yes | - |
| Colored Console | EX | Yes | - | - |
| Console | Yes | Yes | Yes | - |
| Database | EX | Yes | Yes | Yes |
| Debug | Yes | Yes | Yes | - |
| Event Log | Yes | Yes | Yes | Yes |
| Event Tracing (ETW) | Yes | - | - | - |
| File | Yes | Yes | Yes | Yes |
| Mail | - | Yes | Yes | Yes |
| Memory | EX | Yes | Yes | - |
| MSMQ | - | - | Yes | Yes |
| Net Send | - | Yes | - | - |
| Remoting | - | Yes | - | - |
| Rolling File | Yes | Yes | Yes | Yes |
| Syslog (unix) | - | Yes | - | - |
| Telnet | - | Yes | - | - |
| UDP | extension | Yes | Yes | - |
| WMI | - | - | - | Yes |
| XML (Service Trace) | Yes | - | - | Yes |
| Forward to .NET Trace | n/a | Limited | Limited | Yes |

Note that for some features the answer is more complex than a simple table, e.g. System.Diagnostics provides some process/thread information in some listeners, with Essential.Diagnostics providing additional information.

In general features are only indicated where they are directly supported by the framework, for example any framework can log exception details as an argument or even just a string, but some have explicit overloads of logging methods with an Exception type parameter.

Similarly, while some explicitly support logging of lambda expressions, any framework with delayed formatting could pass in a wrapper object that evaluates a lambda when ToString() is called.

## Performance comparison

See the solution in the Performance folder of the source code for the test harness used for comparing performance.

A comparison of logging frameworks should also compare the overhead of the different frameworks, i.e. the overhead of adding trace messages and then ignoring or filtering them out (which should be the majority situation).

All the frameworks allow you to selectively turn on sections of messages to control the volume captured. Whilst the efficiency of capturing the messages you do want may be important, it is the overhead of ignoring the ones you don’t want that is generally the main concern.

The following table shows the results of some performance testing under the following scenarios:

\* All messages ignored, i.e. tracing turned off.

\* One source turned on at Warning level, capturing 16 messages per million trace statements.

\* All sources turned on at Warning level, capturing 245 messages per million trace statements.

\* One source turned to full (all messages), with others at Warning level, capturing 3,907 messages per million trace statements.

The source code for the performance testing application is available in the code repository if you would like to run the tests yourself. Note that the absolute values will change depending on the system they are run on.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Base test framework (ms) | | | | | | 56 |
| Logging overhead (1 million log messages): | | | | | | |
|  | System. Diagnostics | System. Diagnostics (not cleared) | System. Diagnostics EX | Log4net | NLog | Enterprise Application Block |
| Logging off | 50 | 50 | 50 | 46 | 3 | > 20,000 |
| Single filtered | 59 | 85 | 54 | 43 | 8 | > 20,000 |
| Multiple filtered | 89 | 93 | 64 | 49 | 61 | > 20,000 |
| One full | 484 | 620 | 238 | 114 | 867 | > 20,000 |

There are also several configurations in the test harness to compare various other scenarios, for example a source with switchValue=”Off” versus no source defined at all.

**Note 1:** Yes, the values for the Enterprise Application Block are what I am getting – around 21-22,000 milliseconds overhead compared to the other frameworks! Now, the EAB does not have delayed formatting, i.e. no overloads that take a format string and arguments, so my test harness does string.Format() for all messages but even removing that it still has 18-19,000 ms overhead.

**Note 2:** With System.Diagnostics to ignore all messages for a source set switchValue="Off" or simply leave out the source altogether. Having a source with no listeners, i.e. using <clear />, has slightly more overhead but isn't too bad.

If you don't clear the list it may look like you have no listeners but instead you will get all messages written to the default listener, which will severely impact the performance of your application. i.e. the worst thing you can do is set the trace switch to allow a lot of messages (e.g. All) without clearing the listeners:

<!-- Bad example: will severely impact performance -->  
 <source name="MySource" switchValue="All">  
 </source>