Autor: Thomas Reinwart

2009-07-29

office@reinwart.com

Inhalts

Logging Möglichkeiten mit .net

Warum ist Logging in einer Applikation wichtig? Einfach um einen Einblick hinter die Fassade der Anwendung zu haben, etwa was macht eigentlich ein Service am Rechner das ständig 50% CPU Zeit benötigt und um Support leisten zu können.
Logging ist mehr als `Console.WriteLine("Log");`. Man kann unter positiv und negativ Logging unterscheiden. Positiv Logging gibt Statusinformation aus, wie etwa „gestartet am“, „Datei xy gelesen“. Diese Ausgaben sollten laufend erfolgen, aber nicht permanent um das System nicht unnötig mit Informationen zu belasten. Zu den positiv Meldungen gehören je nach eingestellten Log Level auch die Log Debug Informationen. Je nach Umfang der eingefügten Log Debugs im Code kann es hier schon zu System Performance beeinflussenden Zuständen kommen, allerdings wird dies nur der Ausnahmezustand im Zuge einer Fehlersuche vorkommen. Zum Negativ Logging gehören natürlich alle Applikationsfehler oder systemkritischen Zustände, die den Betrieb der eigenen Applikation nicht möglich machen.
Die Logging Information selber muss aufschlussreich sein. Also nicht „Ein Fehler ist aufgetreten“ oder „Datei kann nicht gelesen werden“ sondern wenn möglich auch die zusammenhängenden Parameter die eine Exception verursachen. Eine sprechende Meldung wäre anstatt „Datei kann nicht gelesen werden“, denn eigentlichen Grund im Exceptionhandling zu erkennen, der sein könnte „Datei xy ist nicht vorhanden“. Ein gutes Logging erspart viel Zeit die Fehler nachzustellen und zu debuggen.
Nicht in ein Logging gehören sensible Daten wie Passwörter, man weiß nie welchen Personen ein Logfile im Supportfall gesendet wird.
Wohin eine Loginformation geschrieben werden soll hängt von der Applikation selber, oder etwa der Unternehmens Policy ab, wie die zahlreichen Anwendungen zentral über deren Log Mechanismen überwacht werden können. (Bsp. Eventlog Überwachen mit Microsoft Operation Manager - MOM)
Das Logging sollte daher außerhalb der eigenen Anwendung konfigurierbar sein, um nicht nachkompilieren und neu ausliefern zu müssen. Der Logvorgang kann so eingerichtet sein, dass es der User merkt oder auch komplett im Hintergrund vor sich geht (Achtung sensible Daten), etwa per Mail oder SMS an den Hersteller. Bei einem Logger stehen mehrere, auch parallele Möglichkeiten der Ausgabe zur Verfügung, auch Filtern der Loglevels ist ein Thema.
Logging muss von Anfang an bei der Implementierung vorgesehen werden und nicht nachträglich an den bereits bekannten kritischen Stellen eingefügt werden.

Natürlich kann man selber auch in Files und Eventlogs loggen. Allerding muss man auch Log Szenarien wie Mandantenfähigkeit oder thread safe Operationen bedenken, um einen Überblick zu bewahren und ohne die eigentliche Applikation auszubremsen.
Über zwei Möglichkeiten bestehender und bewährter Logging Komponenten für .net möchte ich hier berichten.

Log4net

Lizenz: Apache License, Version 2.0

Log4net besteht aus der Basis Komponente und den Appendern. Ein Log Appender stellt eine spezielle Log Funktion dar. Die Konfiguration ist XML basierend.

Über die externe XML Konfiguration können die Log Appender und ihre Eigenschaften zur Laufzeit angepasst werden, ohne dass die eigene Applikation neu gestartet wird. D.h. ich kann die Logging Level eines Appenders verändern - etwa den Loglevel von Error auf All stellen, um einen Fehler auf die Spur zu kommen. Beim Instanziieren wird log4net das Configfile bekanntgegeben, über den Filewatcher erkennt es die Änderungen am File und verwendet die aktuelle Config.
Die Logger Konfiguration muss nicht in der App.Config bzw. Web.Config definiert werden, dies kann bzw. mach sogar Sinn es in einem eigenem XML File zu hinterlegen. Wenn es in der Web.Config steht, würde beim Speichern der Web.Config die IIS Session restartet werden.

```
XmlConfigurator.ConfigureAndWatch(configfilename);
```

Auch das Log Layout kann zu jedem Appender konfiguriert werden.

```
Bsp:
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%d [%t] %-5p %c [%x] - %m%n " />
      </layout>
```

Log Ausgabe:
2009-06-03 07:33:36,888 [3004] INFO DefaultLogging – Beispiel Logausgabe

Unterstützt die Frameworks:

- Microsoft .NET Framework 1.0 (1.0.3705)
- Microsoft .NET Framework 1.1 (1.1.4322)
- Microsoft .NET Framework 2.0 (2.0.50727)
- Microsoft .NET Compact Framework 1.0
- Mono 1.0
- Mono 2.0
- Microsoft Shared Source CLI 1.0
- CLI 1.0 Compatible

In der eigenen Applikation werden an passenden Stellen im Code Log Aufrufen hinzugefügt.

Bsp:

```
Log.Info("Datei wird geladen");

Log.Warn(string.Format("Die Datei {0} kann nicht gefunden werden",fileinfo));

catch (Exception ex)
{
    Log.Error(ex);
}
```

Logging Möglichkeiten in .net

Log Core Levels:

Level	Nutzen wenn
ALL	Gibt immer alles aus, Level unabhängig.
DEBUG	Entwickler Infos, Fehlersuche.
INFO	Positiv Logging
WARN	Warnungen, Komponente kann aber weiterarbeiten
ERROR	Fehler, unhandled Exceptions
FATAL	Fatale Fehler
OFF	Logging komplett abdrehen

Log Appender

Appender	Log Event Ausgabe in Form von
AdoNetAppender	Datenbanklogging (MS SQL Server, MS Access, Oracle, IBM DB2, SQLite, …)
AnsiColorTerminalAppender	ANSI Terminal
AspNetTraceAppender	asp Seite
ColoredConsoleAppender	Farbe auf Console (Dos Fenster)
ConsoleAppender	Console
EventLogAppender	Windows Eventlog
FileAppender	Dateilogging
LocalSyslogAppender	Syslog Service (Unix)
MemoryAppender	Memory Buffer
NetSendAppender	Windows Messenger Service
OutputDebugStringAppender	Debugger
RemoteSyslogAppender	Remote Syslog Service (UDP network)
RemotingAppender	.net remoting
RollingFileAppender	Rotierendes Dateilogging, konfigurierbar Anzahl Files und Größe
SmtpAppender	SMTP Email Versand
TelnetAppender	Telnet
TraceAppender	.net trace
UdpAppender	UDP datagrams über UDP Client

Es können auch eigene Log Appender erstellt werden, etwa einen SMS Appender.

Log4net Beispiel Konfiguration

Beispiel und Beschreibung einer Log Konfiguration:

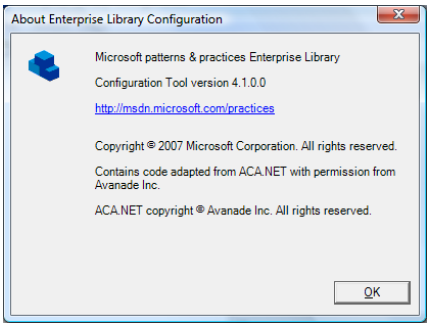
Im `<root>` wurde in diesem Beispiel festgelegt, dass es alle (`ALL`) Events betrifft die an den `LogRollingFileAppender` und an den `ForwardingAppender` weitergeleitet werden.
Im `LogRollingFileAppender` wird alles entsprechend dieser Appender Konfiguration geloggt.
Im `ForwardingAppender` festgelegt, dass alle Log Events ab inklusive dem Level `WARN` (Warn, Error und Fatal) alle Events an den `SmtpAppender` weitergeleitet werden.
Wie man erkennen kann, lässt sich hier mit eine Log Level / Appender Hierarchie erstellen.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <!-- <section name = "log4net"  type="log4net.Config.Log4NetConfigurationSectionHandler,
log4net-net-1.1" /> -->
    <section name="log4net" type="System.Configuration.IgnoreSectionHandler" />
  </configSections>

  <!-- This section contains the log4net configuration settings -->
  <log4net>
    <!-- Define some output appenders -->
    <appender name="ForwardingAppender" type="log4net.Appender.ForwardingAppender" >
      <threshold value="WARN"/>
      <appender-ref ref="SmtpAppender" />
    </appender>
    <!-- Setup the root category, add the appenders and set the default level -->
    <root>
      <level value="ALL" />
      <appender-ref ref="LogRollingFileAppender" />
      <appender-ref ref="ForwardingAppender" />
    </root>
    <!-- Define some output appenders -->
    <appender name="LogRollingFileAppender" type="log4net.Appender.RollingFileAppender">
      <param name="File" value="c:\temp\My_UnitTest.log" />
      <param name="AppendToFile" value="true" />
      <param name="MaxSizeRollBackups" value="9" />
      <param name="MaximumFileSize" value="10MB" />
      <param name="RollingStyle" value="Size" />
      <param name="StaticLogFileName" value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <param name="Header" value="[Start]\r\n" />
        <param name="Footer" value="[End]\r\n" />
        <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] - %m%n" />
      </layout>
    </appender>
    <appender name="LogFileAppender" type="log4net.Appender.FileAppender" >
      <file value="c:\temp\My_UnitTest.log" />
      <appendToFile value="true" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%n%d [%t] %-5p %c - %m" />
      </layout>
    </appender>
    <appender name="SmtpAppender" type="log4net.Appender.SmtpAppender">
      <to value= "" />
      <from value= "" />
      <subject value= "Logger" />
      <smtpHost value= "" />
      <bufferSize value= "20" />
      <!-- records -->
      <lossy value= "false" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%n%date [%t] %-5p %c - %m" />
      </layout>
    </appender>

  </log4net>
</configuration>
```

Die XML Konfiguration ist überschaubar einfach. Ein GUI Konfiguration gibt es von log4net (Apache.org) selber nicht, aber im Internet findet man Editoren für log4net.

Logging Möglichkeiten in .net

Microsoft Enterprise Library Logging



Lizenz: Microsoft Public License (Ms-PL)

Voraussetzungen:

- Microsoft Windows XP Professional, Windows Server 2003, Windows Server 2008, oder Windows Vista Betriebssystem
- Microsoft .NET Framework 3.5 oder höher
- Microsoft Visual Studio 2008 Entwicklungsumgebung

Log Möglichkeiten	Log Event Ausgabe in Form von
Database Trace Listener	Datenbank
Email TraceListener	SMTP Email
FlatFile TraceListener	Single file
Formatted EventLog TraceListener	Eventlog
Msmq TraceListener	Microsoft Message Queue
Rolling Flat File Trace Listener	Rolling file
System.Diagnostics TraceListener	Event
WMI TraceListener	WMI
XML Trace Listener	XML

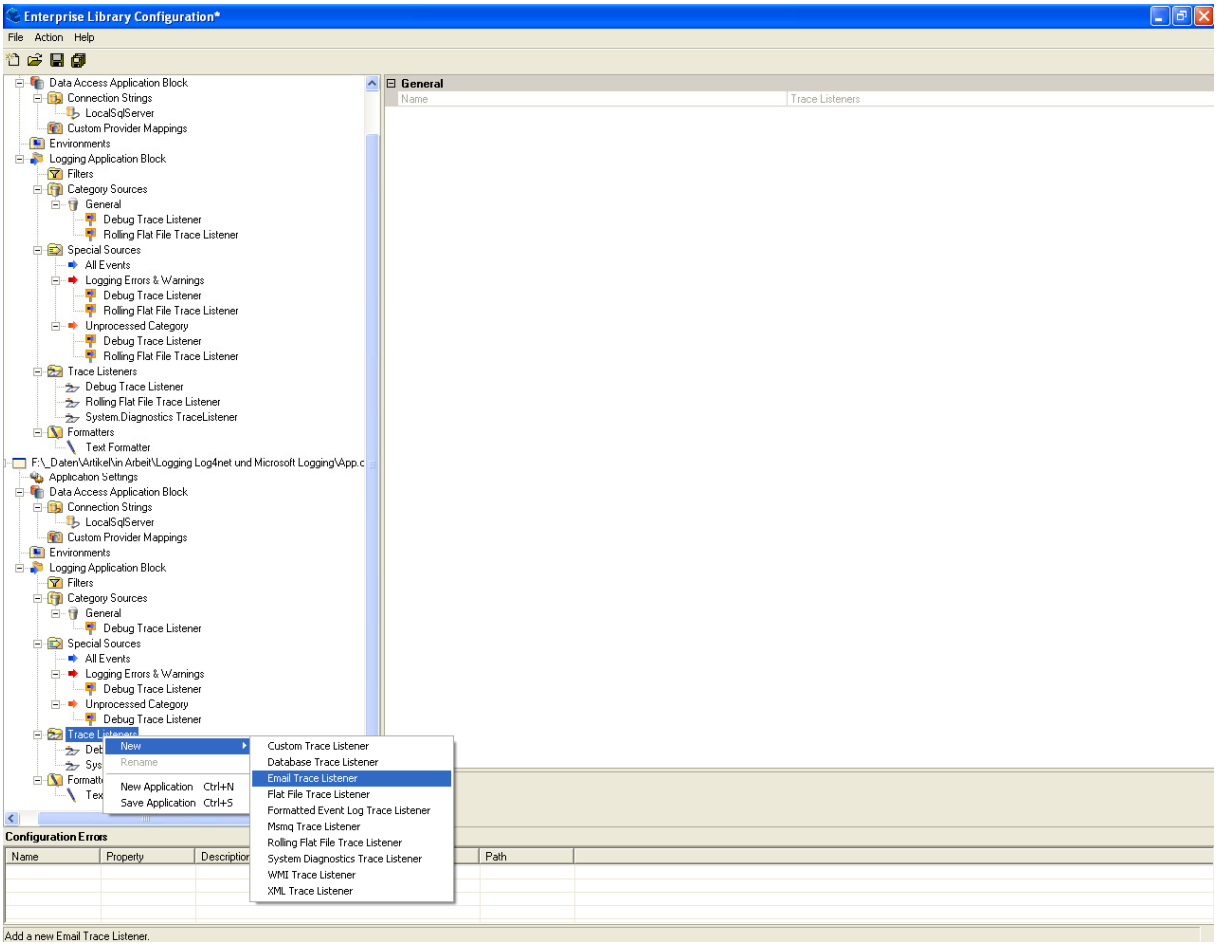
Beispiel Konfiguration

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <loggingConfiguration name="Logging Application Block" tracingEnabled="true"
    defaultCategory="General" logWarningsWhenNoCategoriesMatch="true">
    <listeners>
      <add
listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.CustomTraceListe
nerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
        traceOutputOptions="LogicalOperationStack" filter="All"
type="LogSample.Tests.Logging.DebugTraceListener, LogSample.Tests, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null"
        name="Debug Trace Listener" initializeData="" formatter="Text Formatter" />
      <add
listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.SystemDiagnostic
sTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
```

```
        traceOutputOptions="LogicalOperationStack" filter="All"
type="System.Diagnostics.ConsoleTraceListener, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
        name="System.Diagnostics TraceListener" initializeData="" />
    </listeners>
    <formatters>
      <add template="{severity} {timestamp} {category} {message}"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
        name="Text Formatter" />
    </formatters>
    <categorySources>
      <add switchValue="All" name="General">
        <listeners>
          <add name="Debug Trace Listener" />
        </listeners>
      </add>
    </categorySources>
    <specialSources>
      <allEvents switchValue="All" name="All Events" />
      <notProcessed switchValue="All" name="Unprocessed Category">
        <listeners>
          <add name="Debug Trace Listener" />
        </listeners>
      </notProcessed>
      <errors switchValue="All" name="Logging Errors &amp;amp; Warnings">
        <listeners>
          <add name="Debug Trace Listener" />
        </listeners>
      </errors>
    </specialSources>
  </loggingConfiguration>
</configuration>
```

Die Konfiguration nach einer ähnlichen Logik wie bei log4net aufgebaut.

Konfiguration mittels GUI
Die Konfiguration kann mittels GUI Unterstützung bearbeitet werden. Es kann mehrere Config Files parallel offen haben. Diverse Listener können einfach über das Kontextmenü hinzugefügt werden, die notwendigen Parameter eingefügt werden. Fehler im Gerüst des Configfiles sind damit ausgeschlossen.



Version History

1.0: 2005/01
1.1: 2005/06: .NET Framework 1.1
2.0: 2006/01: .NET Framework 2.0
3.0: 2007/04: .NET Framework 3.0
3.1: 2007/05: kleinere Erweiterungen und Policy injection application blocks
4.0: 2008/05: Unity Application Blocks, Visual Studio 2008
4.1: 2008/10: Aktuelle Version für .net 3.5 und Visual Studio 2008 Sp1
5.0: 2010 ?

Links

Enterprise Library 4.1 – October 2008 (for .NET Framework 3.5 and Visual Studio 2008)
http://msdn.microsoft.com/en-us/library/dd203099.aspx

Logging Möglichkeiten in .net