Ablaufprotokollierung (3)

Die ersten beiden Teile der Reihe haben die Grundlagen zur Ablaufprotokollierung behandelt. Trace, Debug, Schalter, Switches und Listener sind jetzt bekannt, ebenso wie das Verhalten der Ablaufverfolgung über die app.config beeinflusst wird. Dabei wurde bislang ausschließlich mit den Bordmitteln des .NET Frameworks gearbeitet.

Wenn nun aber die Standardausgabe nicht ausreicht, ist es Zeit selbst Hand anzulegen und einen eigenen TraceListener zu schreiben! Dies ist das Thema des dritten Teils dieser Reihe.

Überblick

Auf wenn das .NET Framework schon einige Listener von Haus aus bereitstellt, so gibt es doch immer Fälle in denen dies nicht ausreicht. Sei es weil eine besondere Funktionalität benötigt wird (z. B. Übergabe an einen Web-Dienst) oder auch nur, um das Ausgabeformat anzupassen.

Im Folgenden soll ein konfigurierbarer Trace-Listener entstehen, der die Ausgabe formatiert in eine Datei schreibt und zudem noch ein paar Komfortfunktionen bietet.

Anforderungen

Folgenden Funktionsumfang wird der TraceListener haben:

- Ausgabe in eine Textdatei
- Konfigurierbarer Dateiname
- Neue Ausgabedatei bei Erreichen einer bestimmten Dateigröße bzw. pro Tag/ Woche/Monat
- Formatierte Ausgabe mit (nahezu) fester Spaltenbreite oder als CSV-Datei

Grundlagen

Bevor es an den Code geht müssen noch einige Abläufe in den Klassen TraceSource und TraceListener betrachtet werden.

Die TraceSource Klasse besitzt die Eigenschaft Listeners in der für ein TraceSource Objekt eine Auflistung der angeschlossenen TraceListener gespeichert wird. Wird im Programm nun z. B. die Methode TraceEvent aufgerufen so wird die Liste der Listener durchlaufen und die entsprechende TraceEvent Methode des jeweiligen Listeners aufgerufen.

In der TraceListener Klasse selbst enden Aufrufe der Methoden TraceEvent, TraceData und TraceTransfer in Aufrufen der (in der Basisklasse abstrakten) Methoden Write und WriteLine. Der Vollständigkeit halber sei noch erwähnt, dass die TraceTransfer Methode wiederum die TraceEvent Methode aufruft.

Ein einfaches Beispiel wird diese Abläufe verdeutlichen:

Dazu erstellen wir zunächst eine neue von TraceListener abgeleitete Klasse. Bei dieser müssen die beiden abstrakten Methoden Write(string message) sowie WriteLine(string message) überladen werden. Die Ausgabe erfolgt im Konsolenfenster; zur Unterscheidung welche Methode die Ausgabe erzeugt hat, wird die Ausgabe unterschiedlich eingefärbt.

```
01 using System;
02 using System.Diagnostics;
03
04 namespace Logging
05 {
```

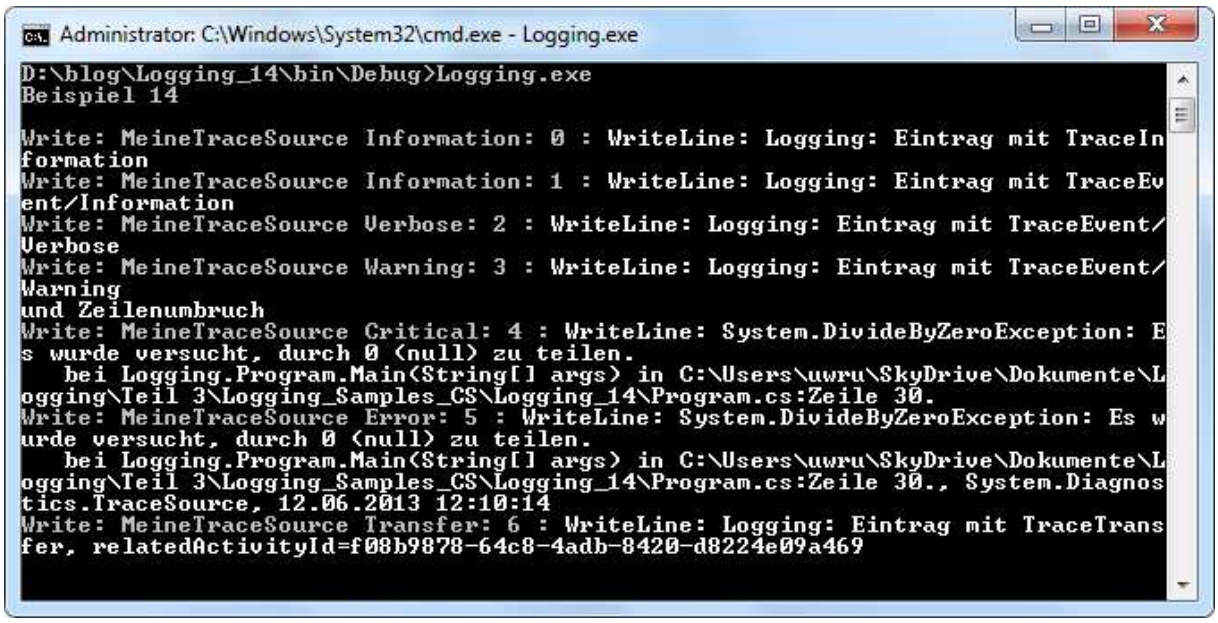
```
06      public class BlogTraceListener : TraceListener
07      {
08          public override void Write(string message)
09          {
10              Console.Write("Write: " + message);
11          }
12
13          public override void WriteLine(string message)
14          {
15              Console.ForegroundColor = ConsoleColor.White;
16              Console.WriteLine("WriteLine: " + message);
17              Console.ForegroundColor = ConsoleColor.Gray;
18          }
19      }
20  }
```

Im Beispielprogramm wird eine Instanz des neuen TraceListeners erstellt und der Listeners Kollektion des TraceSource Objekts hinzugefügt. Anschließend werden über die TraceSource einige Trace Nachrichten ausgegeben.

```
01  using System;
02  using System.Diagnostics;
03
04  namespace Logging
05  {
06      class Program
07      {
08          static void Main(string[] args)
09          {
10              Console.WriteLine("Beispiel 14");
11              Console.WriteLine("");
12
13              SourceSwitch loggingSwitch = new SourceSwitch("testSwitch");
14              loggingSwitch.Level = SourceLevels.All;
15              BlogTraceListener listener = new BlogTraceListener();
16
17              TraceSource loggingSource = new TraceSource("MeineTraceSource");
18
19              loggingSource.Switch = loggingSwitch;
20              loggingSource.Listeners.Add(listener);
21
22              loggingSource.TraceInformation("Logging: Eintrag mit
    TraceInformation");
23              loggingSource.TraceEvent(TraceEventType.Information, 1, "Logging:
    Eintrag mit TraceEvent/Information");
24              loggingSource.TraceEvent(TraceEventType.Verbose, 2, "Logging: Eintrag
    mit TraceEvent/Verbose");
25              loggingSource.TraceEvent(TraceEventType.Warning, 3, "Logging: Eintrag
    mit TraceEvent/Warning" + Environment.NewLine + "und Zeilenumbruch");
26
27              try
28              {
29                  int i = 0;
30                  int r = 5 / i;
31              }
32              catch (Exception ex)
33              {
34                  loggingSource.TraceData(TraceEventType.Critical, 4, ex);
35                  loggingSource.TraceData(TraceEventType.Error, 5, new object[]
    {ex, loggingSource, new DateTime(DateTime.Now.Ticks) } );
36              }
37
38              loggingSource.TraceTransfer(6, "Logging: Eintrag mit TraceTransfer",
    Guid.NewGuid());
39
```

```
40              loggingSource.Close();
41
42              Console.ReadLine();
43          }
44      }
45  }
```

Wird das Beispielprogramm ausgeführt, erhält man folgende Ausgabe:



Dabei fällt auf, dass die Ausgaben stets durch einen Aufruf von Write mit Ablaufverfolgungsinformationen (Event Typ und ID) sowie einem Aufruf von WriteLine mit der eigentlichen Nachricht erfolgen.

Daraus folgt, dass allein mit dem Überschreiben der Write/WriteLine Methoden die gewünschte Funktionalität nicht zu erreichen ist, sondern dass auch TraceData, TraceEvent und TraceTransfer angepasst werden müssen.

Die app.config Datei

```
01  <?xml version="1.0" encoding="utf-8" ?>
02  <configuration>
03    <system.diagnostics>
04      <sources>
05        <source name="MeineTraceSource" switchName="MeinSchalter">
06          <listeners>
07            <add name="blogTraceListener" />
08            <remove name="Default" />
09          </listeners>
10        </source>
11      </sources>
12      <sharedListeners>
13        <add name="blogTraceListener" type="Logging.BlogTraceListener, Logging"
    initializeData="C:\Temp\LogFile-{0}-{1}-{2:D8}-{3:D8}.log;T;FIX" />
14      </sharedListeners>
15      <switches>
16        <add name="MeinSchalter" value="Information,ActivityTracing" />
17      </switches>
18    </system.diagnostics>
19  </configuration>
```

Der neue Listener wird wie üblich im Abschnitt <sharedListeners> eingebunden. Auch wenn die Parameter schon durch die Standard Listener bekannt sind, werfen wir diesmal einen genaueren Blick auf sie:

`name="blogTraceListener"`:
Wie bei allen anderen Listenern - das Kind braucht einen Namen.

`type="Logging.BlogTraceListener, Logging"`:
Hier muss der vollständig gekennzeichnete Typname den Listeners angegeben werden.
Hinweis: Für alle Beispielprojekte wurde in den Projekteigenschaften für Assemblyname und Standardnamespace Logging eingetragen.

`initializeData="C:\Temp\LogFile-{0}-{1}-{2:D8}-{3:D8}.log;T;FIX"`:
Die hier angegebene Zeichenkette wird an den Konstruktor des Listeners übergeben.

Die BlogTraceListener-Klasse

Um also den neuen Listener mit den angegebenen Daten zu initialisieren muss der Konstruktor *TraceListener(string name)* implementiert werden.

```
01 public BlogTraceListener(string initializeData)
02 {
03     initData(initializeData);
04 }
```

In der Variable *initializeData* wird genau der String-Wert übergeben, der in der *app.config* angegeben wurde; im Beispiel oben also "*C:\Temp\LogFile-{0}-{1}-{2:D8}-{3:D8}.csv;T;EXCSV*". *initData* verarbeitet dann diesen String, um Variablen zu initialisieren, den Dateinamen aufzubauen, erste Informationen in die Log-Datei zu schreiben, etc. Details dazu können den Kommentaren im Quellcode entnommen werden.

Die formatierte Ausgabe in die Protokolldatei erfolgt ausschließlich mit der nachfolgenden Methode writeLine().

```
01 private void writeLine(string message, string category, long id, DateTime
   dateTime, int threadId, int processId)
02 {
03     fileOpen();
04
05     checkSwitchReason();
06
07     if (category.Length > maxCategoryNameLength)
08         category = category.Substring(0, maxCategoryNameLength);
09
10     string formatString;
11     if (outputFormat == OutputFormat.CSV || outputFormat ==
   OutputFormat.ExcelCSV)
12     {
13         if (outputFormat == OutputFormat.ExcelCSV)
14             formatString = messageFormatStringCSV.Replace(",", ";");
15         else
16             formatString = messageFormatStringCSV;
17
18         category = string.Format("\"{0}\"", category);
19         if (message.IndexOfAny(new char[] { '"', ',', ';' }) != -1)
20             message = message.Replace("\"", "\"\"");
21         message = string.Format("\"{0}\"", message);
22     }
23     else
24         formatString = messageFormatStringFixedFirstLine;
25
26     if (outputFormat == OutputFormat.CSV || outputFormat ==
```

```
OutputFormat.ExcelCSV)
    {
        logFileStream.WriteLine(formatString,
            category.ToUpper(),
            id,
            threadId,
            processId,
            dateTime.ToString("yyyy-MM-dd HH:mm:ss"),
            message);
    }
    else
    {
        string[] lines = message.Split(new string[] { Environment.NewLine },
StringSplitOptions.None);
        bool firstLine = true;

        foreach (string line in lines)
        {
            if (firstLine)
            {
                logFileStream.WriteLine(formatString,
                    category.ToUpper(),
                    id,
                    threadId,
                    processId,
                    dateTime.ToString("yyyy-MM-dd HH:mm:ss"),
                    line);
                firstLine = false;
            }
            else
                logFileStream.WriteLine(messageFormatStringFixedLines, line);
        }
    }

    loggedMessages++;

    fileClose();
}
```

Der Ablauf ist denkbar einfach:

1. Protokolldatei öffnen
2. Prüfen, ob die Protokolldatei gewechselt werden soll - wenn ja:
    a. Informationen zum Wechsel in die Protokolldatei schreiben
    b. Datei schließen und verschieben
    c. Neue Protokolldatei öffnen
3. Je nach Ausgabeformat die Information in die Protokolldatei schreiben
4. Protokolldatei schließen

Nun zum "eigentlichen" Protokollieren, d. h. zu Write/WriteLine und den verschiedenen Trace Methoden. Da ja ausschließlich writeLine() für die Ausgabe in die Datei verantwortlich ist, erfolgt in diese Methoden ein entsprechender Aufruf.

Write

Da eine der Anforderungen an den Listener ist, dass jeder Trace Eintrag in einer eigenen Zeile steht, werden sämtliche Write-Methoden so überschrieben, dass sie das entsprechende WriteLine Pendant aufrufen:

```
public override void Write(string message)
{
    this.WriteLine(message);
}

```

```
06  public override void Write(string message, string category)
07  {
08      this.WriteLine(message, category);
09  }
10
11  public override void Write(object o)
12  {
13      this.WriteLine(o);
14  }
15
16  public override void Write(object o, string category)
17  {
18      this.WriteLine(o, category);
19  }
```

WriteLine

Der Aufruf von writeLine() erfolgt in der Methode WriteLine(string message, string category). Die anderen WriteLine-Methoden rufen eben diese Methode auf.

```
01  public override void WriteLine(string message)
02  {
03      this.WriteLine(message, defaultCategory);
04  }
05
06  public override void WriteLine(string message, string category)
07  {
08      int threadId = Thread.CurrentThread.ManagedThreadId;
09      int processId = Process.GetCurrentProcess().Id;
10
11      this.writeLine(message, category, 0, DateTime.Now, threadId, processId);
12  }
13
14  public override void WriteLine(object o)
15  {
16      this.WriteLine(o, defaultCategory);
17  }
18
19  public override void WriteLine(object o, string category)
20  {
21      this.WriteLine(o.ToString(), category);
22  }
```

TraceData, TraceEvent und TraceTransfer

Wie weiter oben schon beschrieben müssen die Methoden überschrieben werden, da die Ausgabe in der Basisklasse über mehrere Write/WriteLine Aufrufe erfolgt und somit nicht die Anforderungen nicht erfüllt. Die überschriebenen Methoden rufen daher direkt writeLine() auf.

```
01  public override void TraceData(TraceEventCache eventCache, string source,
    TraceEventType eventType, int id, object data)
02  {
03      this.writeLine(source + ": " + data.ToString(), eventType.ToString(), id,
    eventCache.DateTime, Convert.ToInt32(eventCache.ThreadId), eventCache.ProcessId);
04  }
05
06  public override void TraceData(TraceEventCache eventCache, string source,
    TraceEventType eventType, int id, params object[] data)
07  {
08      string message = "";
09
10      foreach (object obj in data)
11      {
12  if (message != "")
```

```
13      message += ", ";
14  message += obj.ToString();
15      }
16      this.writeLine(source + ": " + message, eventType.ToString(), id,
    eventCache.DateTime, Convert.ToInt32(eventCache.ThreadId), eventCache.ProcessId);
17  }
18
19  public override void TraceEvent(TraceEventCache eventCache, string source,
    TraceEventType eventType, int id)
20  {
21      this.TraceEvent(eventCache, source, eventType, id, "");
22  }
23
24  public override void TraceEvent(TraceEventCache eventCache, string source,
    TraceEventType eventType, int id, string message)
25  {
26      if ((eventType == TraceEventType.Error || eventType ==
    TraceEventType.Critical) && eventCache.Callstack != "")
27      {
28  if (message != "")
29      message += Environment.NewLine;
30  message += eventCache.Callstack;
31      }
32      this.writeLine(source + ": " + message, eventType.ToString(), id,
    eventCache.DateTime, Convert.ToInt32(eventCache.ThreadId), eventCache.ProcessId);
33  }
34
35  public override void TraceEvent(TraceEventCache eventCache, string source,
    TraceEventType eventType, int id, string format, params object[] args)
36  {
37      if (args != null)
38  this.TraceEvent(eventCache, source, eventType, id, string.Format(format, args));
39      else
40  this.TraceEvent(eventCache, source, eventType, id, format);
41  }
42
43  public override void TraceTransfer(TraceEventCache eventCache, string source, int
    id, string message, Guid relatedActivityId)
44  {
45      TraceEventType eventType = TraceEventType.Transfer;
46      this.writeLine(source + ": " + relatedActivityId.ToString() + ": " + message,
    eventType.ToString(), id, eventCache.DateTime,
    Convert.ToInt32(eventCache.ThreadId), eventCache.ProcessId);
47  }
```

Der BlogTraceListener im Einsatz

Im folgenden Beispielprogramm kommt der neue Listener dann zum Einsatz:

```
01  using System;
02  using System.Diagnostics;
03
04  namespace Logging
05  {
06      class Program
07      {
08          static void Main(string[] args)
09          {
10              Console.WriteLine("Beispiel 15");
11              Console.WriteLine("");
12
13              TraceSource loggingSource = new TraceSource("MeineTraceSource");
14
15              loggingSource.TraceInformation("Logging: Eintrag mit
```

```
TraceInformation");
            loggingSource.TraceInformation("Logging: Eintrag mit
TraceInformation");
            loggingSource.TraceEvent(TraceEventType.Information, 1, "Logging:
Eintrag mit TraceEvent/Information");
            loggingSource.TraceEvent(TraceEventType.Verbose, 2, "Logging: Eintrag
mit TraceEvent/Verbose");
            loggingSource.TraceEvent(TraceEventType.Warning, 3, "Logging: Eintrag
mit TraceEvent/Warning" + Environment.NewLine + "und Zeilenumbruch");

            try
            {
                int i = 0;
                int r = 5 / i;
            }
            catch (Exception ex)
            {
                loggingSource.TraceData(TraceEventType.Critical, 4, ex);
                loggingSource.TraceData(TraceEventType.Error, 5, new object[] {
ex, loggingSource, new DateTime(DateTime.Now.Ticks) });
            }

            loggingSource.TraceTransfer(6, "Logging: Eintrag mit TraceTransfer",
Guid.NewGuid());

            loggingSource.Close();

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

Lässt man das Programm laufen, findet man im Verzeichnis *C:\Temp* die Protokolldatei. Diese sieht dann in etwa wie folgt aus:

```
TRACE        0000 00010 04336 2013-05-08 16:34:32 Trace gestartet - DebugVersion
TRACE        0000 00010 04336 2013-05-08 16:34:32 initializeData="C:\Temp\LogFile-{0}-
{1}-{2:D8}-{3:D8}.log;T,FIX"
TRACE        0000 00010 04336 2013-05-08 16:34:32 Dateiwechsel täglich
TRACE        0000 00010 04336 2013-05-08 16:34:32 Assembly:  1.0.0.0
TRACE        0000 00010 04336 2013-05-08 16:34:32 Framework: 4.0.30319.18034
TRACE        0000 00010 04336 2013-05-08 16:34:32 Anwender:  urunkel
TRACE        0000 00010 04336 2013-05-08 16:34:32 Domain:    RUNKEL
TRACE        0000 00010 04336 2013-05-08 16:34:32 Hostname:  VM-DEV
TRACE        0000 00010 04336 2013-05-08 16:34:32 ------------------------------------
----------------------------------
TRACE        0000 00010 04336 2013-05-08 16:34:32 Thread:    10
TRACE        0000 00010 04336 2013-05-08 16:34:32 Process:   4336
TRACE        0000 00010 04336 2013-05-08 16:34:32 Session:   1
TRACE        0000 00010 04336 2013-05-08 16:34:32 ------------------------------------
----------------------------------
TRACE        0000 00010 04336 2013-05-08 16:34:32
INFORMATION  0000 00010 04336 2013-05-08 14:34:32 MeineTraceSource: Logging: Eintrag
mit TraceInformation
INFORMATION  0000 00010 04336 2013-05-08 14:34:32 MeineTraceSource: Logging: Eintrag
mit TraceInformation
INFORMATION  0001 00010 04336 2013-05-08 14:34:32 MeineTraceSource: Logging: Eintrag
mit TraceEvent/Information
WARNING      0003 00010 04336 2013-05-08 14:34:32 MeineTraceSource: Logging: Eintrag
mit TraceEvent/Warning
                                                  und Zeilenumbruch
CRITICAL     0004 00010 04336 2013-05-08 14:34:32 MeineTraceSource:
System.DivideByZeroException: Es wurde versucht, durch 0 (null) zu teilen.
                                                     bei Logging.Program.Main(String[]
args) in C:\Users\uwru\SkyDrive\Dokumente\Logging\Teil
```

```
3\Logging_Samples_CS\Logging_15\Program.cs:Zeile 24.
ERROR        0005 00010 04336 2013-05-08 14:34:32 MeineTraceSource:
System.DivideByZeroException: Es wurde versucht, durch 0 (null) zu teilen.
                                                    bei Logging.Program.Main(String[]
args) in C:\Users\uwru\SkyDrive\Dokumente\Logging\Teil
3\Logging_Samples_CS\Logging_15\Program.cs:Zeile 24., System.Diagnostics.TraceSource,
08.05.2013 16:34:32
TRANSFER     0006 00010 04336 2013-05-08 14:34:32 MeineTraceSource: 73c0943a-4e5a-47dc-
8436-e999b8b9d92e: Logging: Eintrag mit TraceTransfer
TRACE        0000 00002 04336 2013-05-08 16:34:34 Trace beendet
```

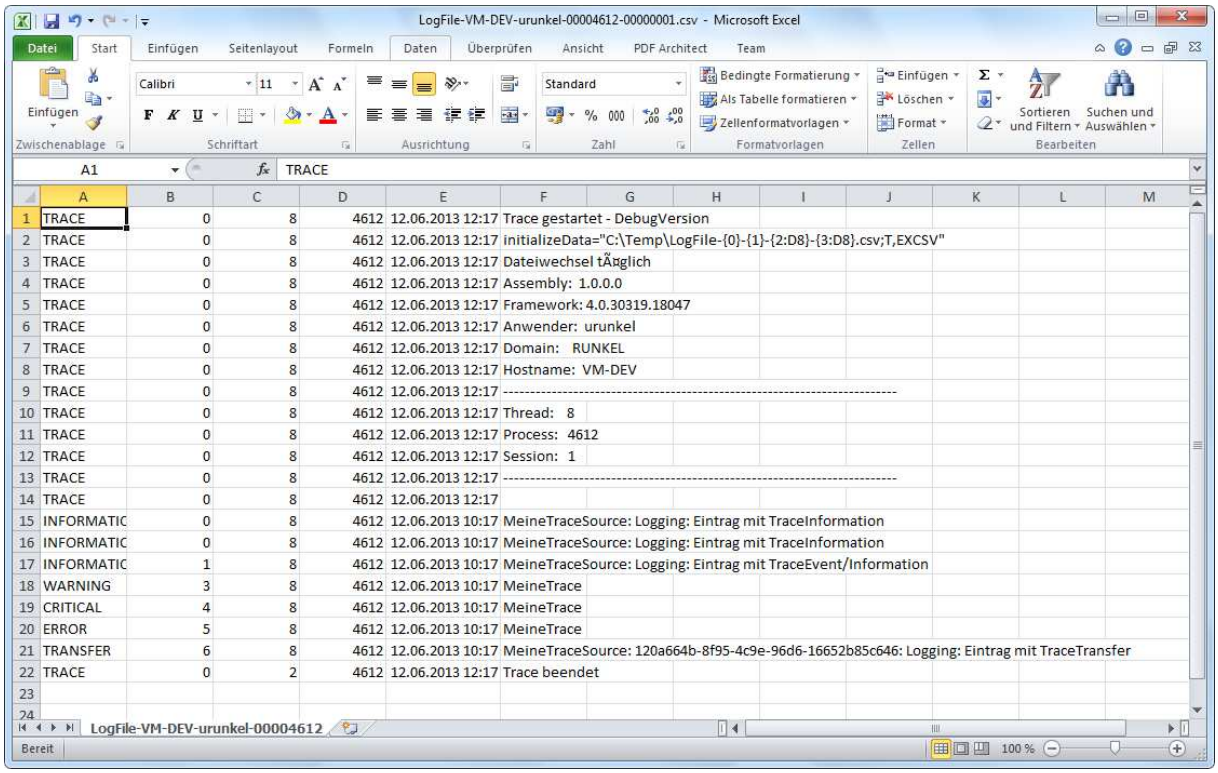
Wir passen nun in der *app.config* den *initializeData* Parameter an

```
12      <sharedListeners>
13        <add name="blogTraceListener" type="Logging.BlogTraceListener, Logging"
    initializeData="C:\Temp\LogFile-{0}-{1}-{2:D8}-{3:D8}.csv;T,EXCSV" />
14      </sharedListeners>
```

und starten das Programm dann noch einmal. Diesmal erhält man folgende Datei:

```
"TRACE";0;11;6712;2013-05-08 16:37:15;"Trace gestartet - DebugVersion"
"TRACE";0;11;6712;2013-05-08 16:37:15;"initializeData=""C:\Temp\LogFile-{0}-{1}-{2:D8}-
{3:D8}.csv;T,EXCSV"""
"TRACE";0;11;6712;2013-05-08 16:37:15;"Dateiwechsel täglich"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Assembly:  1.0.0.0"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Framework: 4.0.30319.18034"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Anwender:  urunkel"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Domain:    RUNKEL"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Hostname:  VM-DEV"
"TRACE";0;11;6712;2013-05-08 16:37:15;"----------------------------------------------
------------------------"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Thread:    11"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Process:   6712"
"TRACE";0;11;6712;2013-05-08 16:37:15;"Session:   1"
"TRACE";0;11;6712;2013-05-08 16:37:15;"----------------------------------------------
------------------------"
"TRACE";0;11;6712;2013-05-08 16:37:15;""
"INFORMATION";0;11;6712;2013-05-08 14:37:15;"MeineTraceSource: Logging: Eintrag mit
TraceInformation"
"INFORMATION";0;11;6712;2013-05-08 14:37:15;"MeineTraceSource: Logging: Eintrag mit
TraceInformation"
"INFORMATION";1;11;6712;2013-05-08 14:37:15;"MeineTraceSource: Logging: Eintrag mit
TraceEvent/Information"
"WARNING";3;11;6712;2013-05-08 14:37:15;"MeineTraceSource: Logging: Eintrag mit
TraceEvent/Warning
und Zeilenumbruch"
"CRITICAL";4;11;6712;2013-05-08 14:37:15;"MeineTraceSource:
System.DivideByZeroException: Es wurde versucht, durch 0 (null) zu teilen.
   bei Logging.Program.Main(String[] args) in D:\dev\blog\Logging\Teil
3\Logging_Samples_CS\Logging_16\Program.cs:Zeile 24."
"ERROR";5;11;6712;2013-05-08 14:37:15;"MeineTraceSource: System.DivideByZeroException:
Es wurde versucht, durch 0 (null) zu teilen.
   bei Logging.Program.Main(String[] args) in D:\dev\blog\Logging\Teil
3\Logging_Samples_CS\Logging_16\Program.cs:Zeile 24., System.Diagnostics.TraceSource,
08.05.2013 16:37:15"
"TRANSFER";6;11;6712;2013-05-08 14:37:15;"MeineTraceSource: 2eb08cc9-0a2f-4236-9943-
9b2ebe30bbcd: Logging: Eintrag mit TraceTransfer"
"TRACE";0;2;6712;2013-05-08 16:37:16;"Trace beendet"
```

Diese CSV-Datei lässt sich dann auch problemlos mit Excel öffnen.



	A	B	C	D	E	F
1	TRACE	0	8	4612	12.06.2013 12:17	Trace gestartet - DebugVersion
2	TRACE	0	8	4612	12.06.2013 12:17	initializeData="C:\Temp\LogFile-{0}-{1}-{2:D8}-{3:D8}.csv;T,EXCSV"
3	TRACE	0	8	4612	12.06.2013 12:17	Dateiwechsel tÃ¤glich
4	TRACE	0	8	4612	12.06.2013 12:17	Assembly: 1.0.0.0
5	TRACE	0	8	4612	12.06.2013 12:17	Framework: 4.0.30319.18047
6	TRACE	0	8	4612	12.06.2013 12:17	Anwender: urunkel
7	TRACE	0	8	4612	12.06.2013 12:17	Domain: RUNKEL
8	TRACE	0	8	4612	12.06.2013 12:17	Hostname: VM-DEV
9	TRACE	0	8	4612	12.06.2013 12:17	----------
10	TRACE	0	8	4612	12.06.2013 12:17	Thread: 8
11	TRACE	0	8	4612	12.06.2013 12:17	Process: 4612
12	TRACE	0	8	4612	12.06.2013 12:17	Session: 1
13	TRACE	0	8	4612	12.06.2013 12:17	----------
14	TRACE	0	8	4612	12.06.2013 12:17	
15	INFORMATIC	0	8	4612	12.06.2013 10:17	MeineTraceSource: Logging: Eintrag mit TraceInformation
16	INFORMATIC	0	8	4612	12.06.2013 10:17	MeineTraceSource: Logging: Eintrag mit TraceInformation
17	INFORMATIC	1	8	4612	12.06.2013 10:17	MeineTraceSource: Logging: Eintrag mit TraceEvent/Information
18	WARNING	3	8	4612	12.06.2013 10:17	MeineTrace
19	CRITICAL	4	8	4612	12.06.2013 10:17	MeineTrace
20	ERROR	5	8	4612	12.06.2013 10:17	MeineTrace
21	TRANSFER	6	8	4612	12.06.2013 10:17	MeineTraceSource: 120a664b-8f95-4c9e-96d6-16652b85c646: Logging: Eintrag mit TraceTransfer
22	TRACE	0	2	4612	12.06.2013 12:17	Trace beendet

Bleibt noch die Frage, wozu auch Write/WriteLine überschreiben, werden diese von TraceData, TraceEvent und TraceTransfer doch gar nicht mehr aufgerufen? Ganz einfach: werden für die Ausgabe Debug oder Trace (statt einer TraceSource) benutzt, ruft beispielsweise Trace.Write() die Write() Methode des Listeners auf.

Wie das dann funktioniert zeigt das nächste Beispiel:

```
01  using System;
02  using System.Diagnostics;
03
04  namespace Logging
05  {
06      class Program
07      {
08          static void Main(string[] args)
09          {
10              BlogTraceListener listener = new
    BlogTraceListener("C:\\Temp\\TraceFile-{0}-{1}-{2:D8}-{3:D8}.log;T,FIX");
11              Trace.Listeners.Add(listener);
12              Trace.Listeners.Remove("Default");
13              Trace.AutoFlush = true;
14
15              Console.WriteLine("Beispiel 17");
16
17              TraceSwitch traceSwitch = new TraceSwitch("MeinSchalter",
    "Ablaufprotokollierung mit Schalter");
18              traceSwitch.Level = TraceLevel.Info;
19
20              Trace.WriteLine("Beispiel 17 gestartet: " + DateTime.Now.ToString());
21              Trace.Write("========================================");
22
23              Trace.WriteLineIf(traceSwitch.TraceError, "Logging: Eintrag mit
    TraceLevel Error");
24              Trace.WriteLineIf(traceSwitch.TraceWarning, "Logging: Eintrag mit
    TraceLevel Warning");
25              Trace.WriteLineIf(traceSwitch.TraceInfo, "Logging: Eintrag mit
    TraceLevel Info");
```

```
26              Trace.WriteLineIf(traceSwitch.TraceVerbose, "Logging: Eintrag mit
    TraceLevel Verbose");
27
28              Console.ReadLine();
29          }
30      }
31  }
```

Die Ausgabe sieht dann wie folgt aus:

```
TRACE        0000 00008 05640 2013-05-08 16:41:18 Trace gestartet - DebugVersion
TRACE        0000 00008 05640 2013-05-08 16:41:18 initializeData="C:\Temp\TraceFile-
{0}-{1}-{2:D8}-{3:D8}.log;T,FIX"
TRACE        0000 00008 05640 2013-05-08 16:41:18 Dateiwechsel täglich
TRACE        0000 00008 05640 2013-05-08 16:41:18 Assembly:  1.0.0.0
TRACE        0000 00008 05640 2013-05-08 16:41:18 Framework: 4.0.30319.18034
TRACE        0000 00008 05640 2013-05-08 16:41:18 Anwender:  urunkel
TRACE        0000 00008 05640 2013-05-08 16:41:18 Domain:    RUNKEL
TRACE        0000 00008 05640 2013-05-08 16:41:18 Hostname:  VM-DEV
TRACE        0000 00008 05640 2013-05-08 16:41:18 ------------------------------------
----------------------------------
TRACE        0000 00008 05640 2013-05-08 16:41:18 Thread:    8
TRACE        0000 00008 05640 2013-05-08 16:41:18 Process:   5640
TRACE        0000 00008 05640 2013-05-08 16:41:18 Session:   1
TRACE        0000 00008 05640 2013-05-08 16:41:18 ------------------------------------
----------------------------------
TRACE        0000 00008 05640 2013-05-08 16:41:18
NONE         0000 00008 05640 2013-05-08 16:41:18 Beispiel 17 gestartet: 08.05.2013
16:41:18
NONE         0000 00008 05640 2013-05-08 16:41:18
========================================
NONE         0000 00008 05640 2013-05-08 16:41:18 Logging: Eintrag mit TraceLevel Error
NONE         0000 00008 05640 2013-05-08 16:41:18 Logging: Eintrag mit TraceLevel
Warning
NONE         0000 00008 05640 2013-05-08 16:41:18 Logging: Eintrag mit TraceLevel Info
TRACE        0000 00002 05640 2013-05-08 16:41:20 Trace beendet
```

Fazit

Neben dem Grundverständnis für die Ablaufprotokollierung in den Teilen Eins und zwei wurde in diesem letzten Teil der Blog-Reihe ein eigener TraceListener erstellt. Dabei soll der hier vorgestellte Listener wiederum nur Grundlage sein. Erweiterte Funktionalitäten wie z. B. Thread-Sicherheit und vieles, vieles mehr wurden dabei nicht berücksichtigt.

Wie so oft lohnt es auch sich etwas umzuschauen, um das Rad nicht neu erfinden zu müssen. Es gibt unzählige und wirklich exzellente (OpenSource) Logging-Frameworks. NLog, log4net, ObjectGuy Framework oder der Logging Application Block der Enterprise Library, um nur einige wenige zu nennen.

Eine Übersicht über zahlreiche Logging Frameworks findet man z. B. hier (http://www.dotnetlogging.com/). Dort gibt es auch einen Vergleich eines kommerziellen (das des Seitenbetreibers) mit einigen der bekanntesten OpenSource Frameworks.