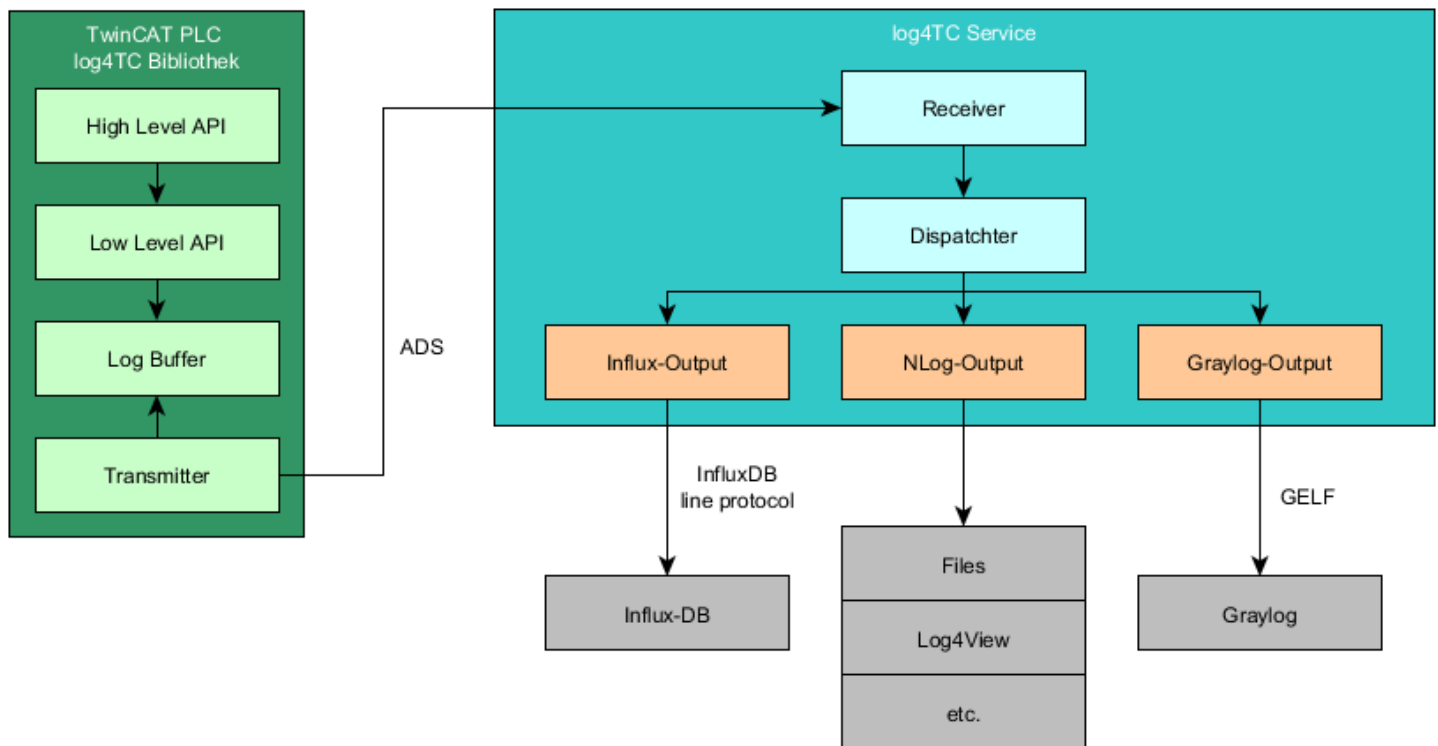


mbc log4TC

Log4TC ist eine Erweiterung für TwinCAT3 von Beckhoff, um direkt aus der SPS Logmeldungen erzeugen zu können. Die Meldungen können transferiert, gefiltert, ausgewertet und an verschiedene Ausgaben weitergeleitet werden.

Log4TC besteht aus zwei Teilen, einer SPS-Bibliothek und einen Windows-Service.

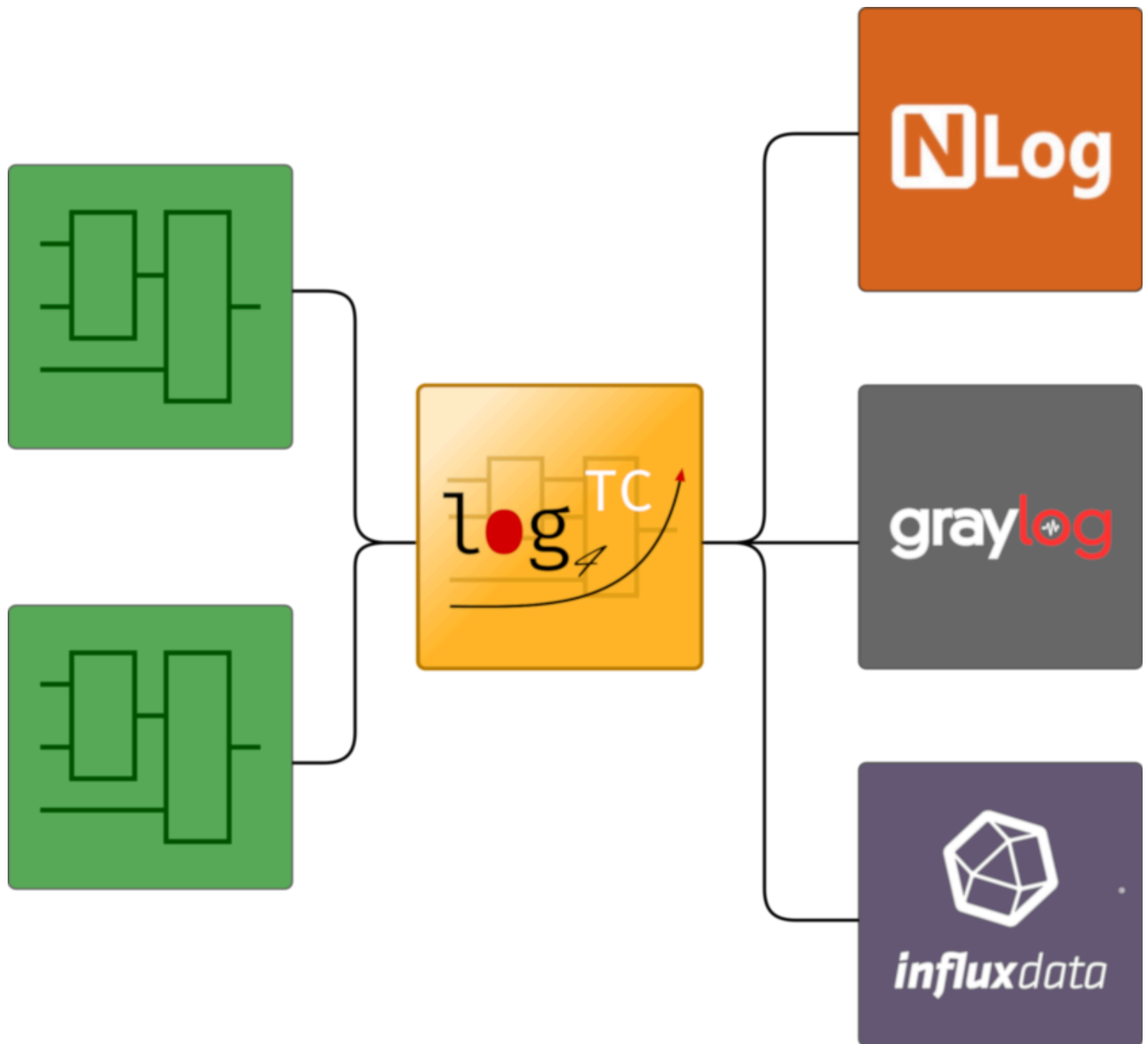


Der log4TC-Service wird normalerweise auf dem Rechner installiert, auf dem auch die SPS läuft, kann aber für bestimmte Einsatzzwecke auch auf einem anderen Rechner für mehrere Steuerung installiert werden.

Features

- Einfache API für die integration in die SPS
- Strukturiertes Logging (<https://message templates.org/>)
- Unterstützung von Context-Eigenschaften auf verschiedenen Ebenen
- Performant und Modular
- Kostenlose Testversion verfügbar
- Lizenzierung über Beckhoff-Mechanismus in Dongle, Klemme oder PC
- Unbegrenzte Ausgabemöglichkeiten (Textdatei, Datenbank, Cloud, usw.)

Ausgaben



Log4TC implementiert Ausgaben über ein Plugin-System. Standardmässig bei der Auslieferung ist die NLog-Ausgabe aktiv. Die Ausgabeplugins werden laufend erweitert, die nächsten Plugins sind Ausgaben für Graylog und InfluxDB.

Bei Bedarf können wir auch kundenspezifische Ausgaben erstellen.

Typische Anwendungsfälle für log4TC

- Fehlertracking und Alarmierung
- Debugging von sporadischen Fehlern ohne Breakpoints
- Ablaufanalysen bei Problemen auch in Nachhinein

- Statistische Auswertungen, z.B. KPI

Nächste Schritte

- [Download](#)[↗]
- [Erste Schritte](#)
- [Referenz](#)

Firewall

Der Log4TC Service erstellt einen eigenen ADS Server auf den die PLC sich verbindet. Der ADS Server läuft auf Port **16150**. Daher muss die Firewall entsprechend konfiguriert werden damit dieser Port erreichbar ist.

Table of Contents

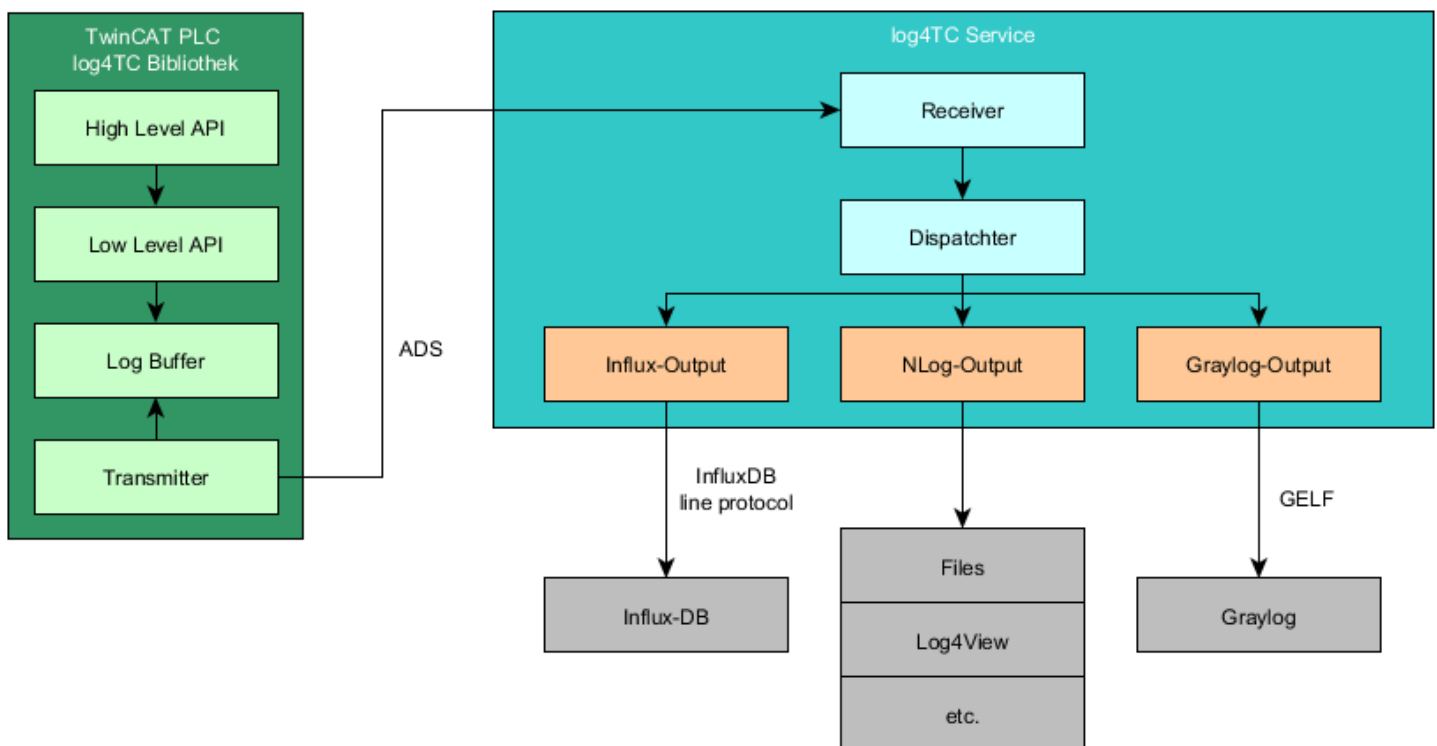
Erste Schritte Tour	5
TwinCat Project anlegen	7
log4TC Library zum SPS-Projekt hinzufügen	9
Ausgabe einer einfachen Log-Meldung	11
Ausgabe von Log-Meldungen mit Argumenten	14
Benutzung von Loggern	16
Integration von Context-Eigenschaften	19
Log-Meldungen mit Log4View beobachten	22
Protokollierung von strukturierten Werten	26

Erste Schritte

Diese geführte Tour stellt die grundlegenden Konzepte und Features von log4TC in einem kleinen zusammenhängenden Projekt vor.

Vorraussetzungen

Damit log4Tc richtig benutzt werden kann, sollte der Aufbau bekannt sein. Es wird dabei unterschieden zwischen den Komponenten **log4TC TwinCat 3 Bibliothek** und dem **log4TC Service**.



Folgende Voraussetzungen haben die beiden Komponenten:

log4TC TwinCat 3 Bibliothek

- [TwinCat 3 \(min. 4022.00\)](#)

log4TC Service

- min Windows 7 SP1 / Windows Embedded Standard 2009
- [ADS Router - TC1000 | TC3 ADSF](#)
- [Verwendet Microsoft .NET 8 \(muss nicht installiert werden\)](#)

Beim [Installieren](#) von log4TC wird eine Default-Konfigurationsdatei mit installiert. Für die nachfolgenden Beispiele wird davon ausgegangen, dass diese Konfiguration aktiv ist

nachfolgenden Beispiele wird davon ausgegangen, dass diese Konfiguration aktiv ist.

Für diese Einführung wird ausserdem vorausgesetzt, dass die SPS und der log4TC auf dem *gleichen* Rechner laufen (TwinCAT Testlizenz ist ausreichend). Dies ist keine Einschränkung von log4TC sondern eine Vereinfachung.

Übersicht

Die Einführung geht schrittweise vor. Es wird empfohlen beim ersten Kontakt mit log4TC alle Schritte nacheinander selbst auszuprobieren.

1. [TwinCAT Projekt anlegen](#)
2. [log4TC-Library hinzufügen](#)
3. [Ausgabe einer einfachen Log-Meldung](#)
4. [Ausgabe von Log-Meldungen mit Argumenten](#)
5. [Benutzung von Loggern](#)
6. [Integration von Context-Eigenschaften](#)
7. [Log-Meldungen mit Log4View beobachten](#)
8. [Protokollierung von strukturierten Werten](#)

Nächster Schritt

[TwinCAT Projekt anlegen](#)

TwinCAT Projekt anlegen

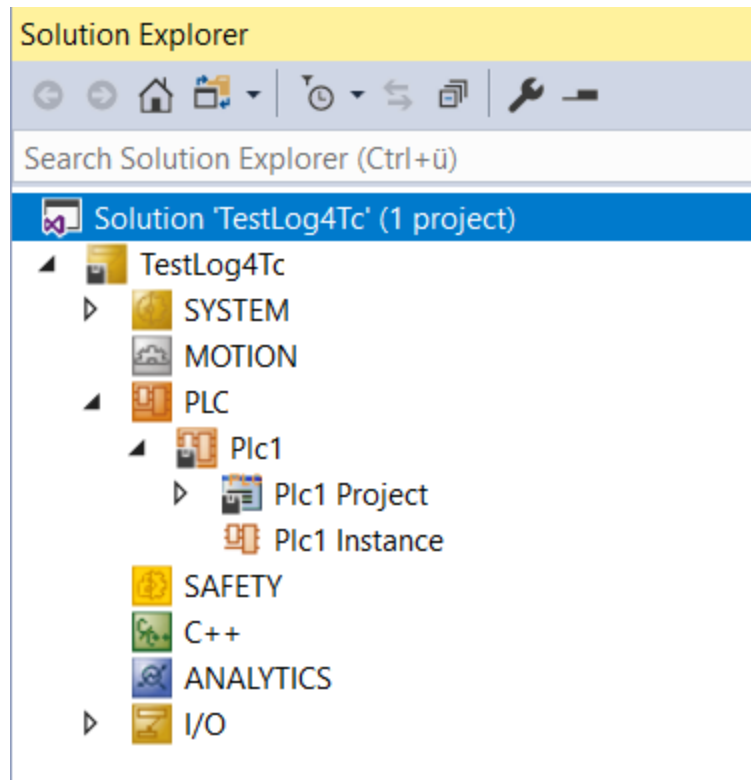
Um den Beispielen zu folgen, wird ein neues TwinCAT Projekt mit einem SPS-Projekt benötigt. Das Projekt muss für das Beispiel auf den gleichen Rechner aktiviert werden, auf dem auch log4TC installiert ist. Eine lokale Testlizenz ist ausreichend.

Solution (TwinCAT Project) anlegen

1. Neues leeres TwinCAT XAE Projekt anlegen
2. Neues "Standard PLC Project" anlegen

Es müssen keine speziellen Einstellungen vorgenommen werden.

Die Visual Studio Solution sollte wie folgt aussehen:



Nächster Schritt

[log4TC-Library hinzufügen](#)

log4TC Library zum SPS-Projekt hinzufügen

Library als Referenz hinzufügen

Die log4TC-Library wird mit der Installation als SPS-Library registriert und muss noch in das Projekt eingebunden werden:

1. Doppelklick auf *Order References* im SPS-Projekt -> *Library Manager* wird geöffnet
2. "Add library" anwählen
3. Im Dialog den Suchtext "log4tc" eingeben; in der Liste sollte die Library "Log4TC" erscheinen
4. Doppelklick auf "Log4TC"

Die Referenzen sollte wie folgt aussehen (Versionnummer kann variieren):

The screenshot shows the Siemens STEP 7 interface. On the left, the 'Solution Explorer' displays a project named 'TestLog4Tc' with a 'References' folder containing 'Log4TC', 'Tc2_Standard', 'Tc2_System', and 'Tc3_Module'. On the right, the 'Library Manager' window is open, showing a table of references:

Name	Namespace	Effective version
Log4TC = Log4TC, * (mbc engineering GmbH)	Log4TC	0.0.1
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH)	Tc2_Standard	3.3.2.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH)	Tc2_System	3.4.21.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH)	Tc3_Module	3.3.21.0

Nächster Schritt

[Ausgabe einer einfachen Log-Meldung](#)

Ausgabe einer einfachen Log-Meldung

Aufruf des Loggers im MAIN

Die in der SPS erzeugten Log-Meldungen werden nicht sofort beim Aufruf eines Log-Bausteins übertragen, sondern werden zunächst in einen Task-spezifischen Puffer gespeichert. Damit diese Meldungen dann an den log4Tc-Service übertragen werden, muss in **jeder** Task das log4Tc aufgerufen werden.

Der Aufruf passiert mit folgenden Code, wir empfehlen diese Anweisung an das Ende jedes Bausteins anzufügen, dass von einer Task referenziert wird, in unseren Fall also im MAIN-Baustein.:

```
PRG_TaskLog.Call();
```

Ausgabe einer Log-Meldung

Im Beispiel soll eine Meldung ausgegeben werden, wenn die SPS startet. TwinCAT stellt ein Flag zur Verfügung, dass im ersten Zyklus auf **TRUE** gesetzt ist:

```
IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN  
    // Hier soll eine Meldung ausgegeben werden  
END_IF
```

Um eine Meldung auszugeben, muss an der markierten Stelle eine Funktion der Log4TC Library aufgerufen werden.

```
F_Log(E_LogLevel.eInfo, 'SPS Task gestartet.');
```

Hinweis: Im Beispiel wird die kurze Form für Bibliotheksaufrufe verwendet. Der Aufruf kann aber auch mit dem Namensraum erfolgen: `log4tc.F_Log(E_LogLevel.eInfo, 'SPS Task gestartet.');`

Der Aufruf besteht aus zwei Parametern:

- **eLogLevel**: Muss immer angegeben werden und definiert den Level der Log-Meldungen. Log4Tc kennt die Stufen Trace, Debug, Info, Warn, Error, Fatal. Weitergehende Informationen zu den Log-Level und ihre Bedeutung sind [hier](#) zu finden.
- **sMessage**: Gibt den Text an der geloggt werden soll.

Der MAIN-Baustein sollte wie folgt aussehen:

PROGRAM MAIN

```
-----  
IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN  
    F_Log(E_LogLevel.eInfo, 'SPS Task gestartet.');
```

END_IF

PRG_TaskLog.Call();

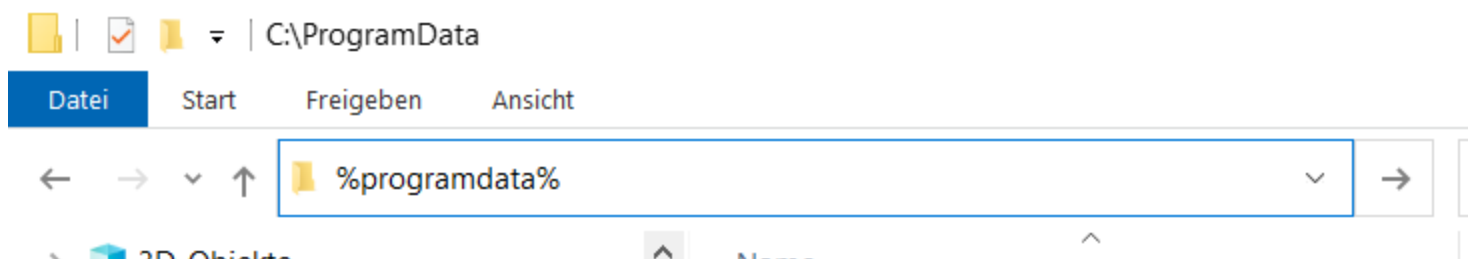
Der Code befindet sich im Beispielprojekt unter den Namen "A_SimpleLogMessage".

Ausführen des SPS-Projekts und anzeige der Meldung

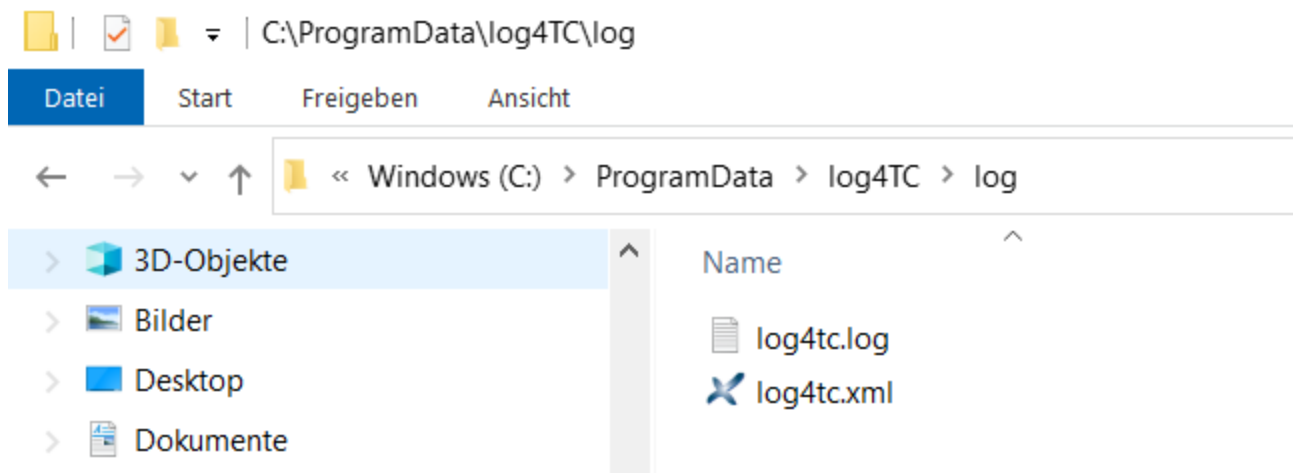
Das Projekt kann es aktiviert, geladen und ausgeführt werden.

Log-Meldungen werden mit der ausgelieferten Konfiguration in das Verzeichnis `%ProgramData%\log4tc\log\` abgelegt.

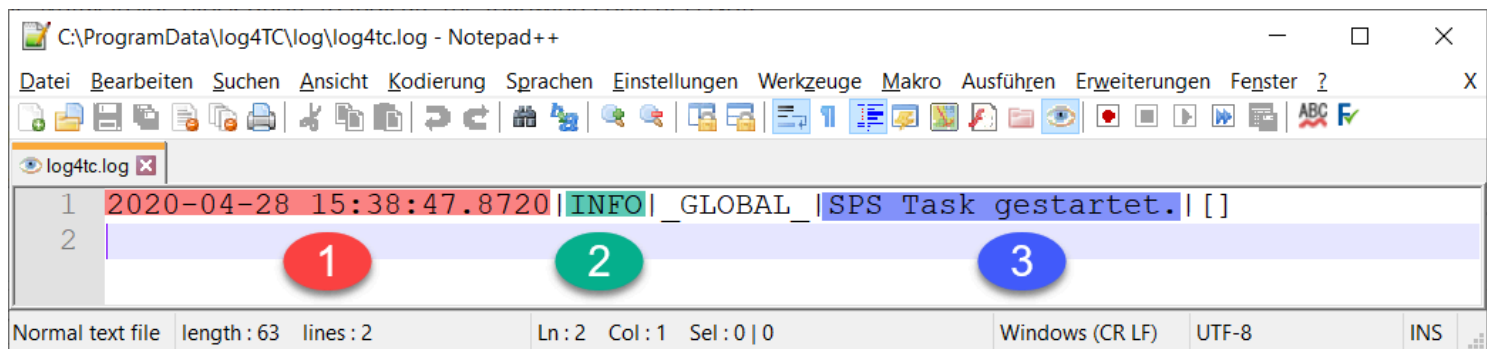
Tip: In Windows ist der Ordner `%ProgramData%` (entspricht normalerweise den Pfad `C:\ProgramData`) versteckt und wird nicht im Explorer. Man kann aber den Text `%programdata%` als Pfad im Explorer eingeben und gelangt dann direkt zum Ordner. Alternativ können auch die Links verwendet werden, die mit der Installation von log4TC im Startmenü angelegt werden.



Im Log-Ordner befinden sich zwei Dateien, momentan geht es nur um die `log4tc.log`.



Die Datei `log4tc.log` kann mit einem normalen Texteditor geöffnet werden (siehe auch [Tools](#)):



Die Log-Meldung besteht aus mehreren Teilen, die durch ein `|`-Zeichen getrennt sind (Das Format einer Meldung kann über die NLog-Konfiguration fast beliebig geändert werden.).

1. Zeitstempel der Meldung (SPS-Zeit) mit 100ns Auflösung (abhängig von Task-Zeit)
2. Log-Level der Meldung, entspricht den ersten Input-Parameter (`E_LogLevel.eInfo`)
3. Meldungstext

Die Erklärung der beiden übrigen Felder (`_GLOBAL` und `[]`) erfolgen später.

Nächster Schritt

[Ausgabe von Log-Meldungen mit Argumenten](#)

Ausgabe von Log-Meldungen mit Argumenten

Log4TC erlaubt es nicht nur konstante Log-Message auszugeben, sondern auch Argumente mitzugeben. Die Argumente stehen dann für den Text zur Formattierung zur Verfügung.

Log4TC stellt Funktionen für 0 bis 10 Argumente zur Verfügung.

Ausgabe von Argumenten mit ANY-Typ

Argumente werden mit dem ST-Datentyp **ANY** übergeben werden. Einzige Voraussetzung dafür ist, dass der Wert als Variable vorliegt. Es können daher keine Literale (z.B. **42**) und keine Ausdrücke (z.B. **nVar * 10**) verwendet werden. Dies ist eine Einschränkung des **ANY**-Typs.

Im Beispiel wird ein Zähler erzeugt, der jede Sekunde um eins hochzählt:

```
VAR
    nCounter          : UINT;
    fbCountTime       : TON := (PT:=T#1S);
END_VAR
-----
fbCountTime(IN:=NOT fbCountTime.Q);
IF fbCountTime.Q THEN
    nCounter := nCounter + 1;
    // Hier Zähler ausgeben
END_IF
```

Als Nächstes wird bei jeder Änderung des Zählers eine Meldung in das Log-File geschrieben. Dazu wird der Logger im **IF/THEN** eingefügt und der aktuellen Zählerwert übergeben:

```
F_LogA1(E_LogLevel.eDebug, 'Zähler geändert, neuer Wert {0}', nCounter);
```

Da ein Argument vom Typ **ANY** übergeben wird, heisst die Funktion **F_LogA1**. Gegenüber **F_Log** besitzt diese einen weiteren Input-Parameter mit dem der aktuelle Zählerwert übergeben wird. Entsprechend würde die Funktion bei zwei Argumenten **F_LogA2** heissen.

Damit der Zähler-Wert auch im Text erscheint, wird ein Platzhalter **{0}** in der Log-Message gesetzt (Platzhalter). Im Beispiel wird dieser Platzhalter bei der Ausgabe durch den Wert der Variable **nCounter** ersetzt.

Der komplette MAIN-Code sieht damit wie folgt aus:

```
PROGRAM MAIN
VAR
    nCounter          : UINT;
    fbCountTime       : TON := (PT:=T#1S);
END_VAR

-----

IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN
    F_Log(E_LogLevel.eInfo, 'SPS Task gestartet. ');
END_IF

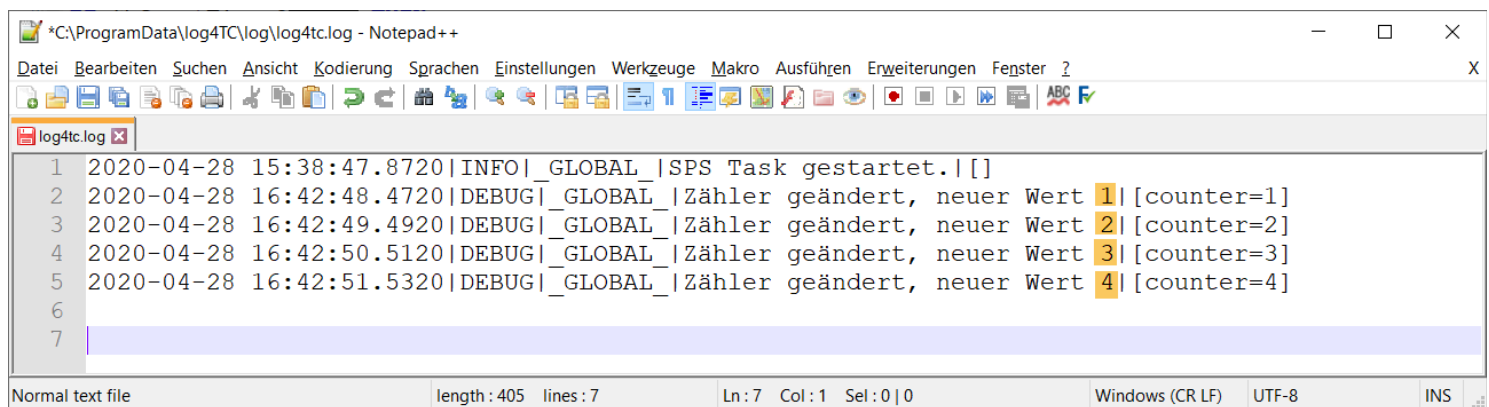
fbCountTime(IN:=NOT fbCountTime.Q);
IF fbCountTime.Q THEN
    nCounter := nCounter + 1;
    F_LogA1(E_LogLevel.eDebug, 'Zähler geändert, neuer Wert {counter}',
nCounter);
END_IF

PRG_TaskLog.Call();
```

Der Code befindet sich im Beispielprojekt unter den Namen "B_LogMessageWithArg".

Aktualisieren des SPS-Codes und beobachten der Log-Meldung

Der geänderte SPS-Code kann jetzt mit einem Online-Change aktualisiert werden. Danach kann man im Log-File jede Sekunde den neuen Zählerwert ablesen.



```
*C:\ProgramData\log4TC\log\log4tc.log - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ? X

log4tc.log X
1 2020-04-28 15:38:47.8720|INFO|_GLOBAL_|SPS Task gestartet. | []
2 2020-04-28 16:42:48.4720|DEBUG|_GLOBAL_|Zähler geändert, neuer Wert 1 | [counter=1]
3 2020-04-28 16:42:49.4920|DEBUG|_GLOBAL_|Zähler geändert, neuer Wert 2 | [counter=2]
4 2020-04-28 16:42:50.5120|DEBUG|_GLOBAL_|Zähler geändert, neuer Wert 3 | [counter=3]
5 2020-04-28 16:42:51.5320|DEBUG|_GLOBAL_|Zähler geändert, neuer Wert 4 | [counter=4]
6
7

Normal text file      length: 405  lines: 7      Ln: 7  Col: 1  Sel: 0 | 0      Windows (CR LF)  UTF-8  INS
```

Nächster Schritt

[Benutzung von Loggern](#)

Benutzung von Loggern

Zweck von *Logger*

Rein technisch betrachtet ist ein *Logger* eine Bezeichnung für eine oder mehrere Log-Meldungen, das durch das gesamte Logging-System weitergereicht wird. Ein *Logger* kann daher z.B. von Filterplugins verwendet werden oder bei der Ausgabe mit geschrieben werden.

Wie genau und ob *Logger* benutzt werden, kann vom Entwickler selbst definiert werden. Bei allen grösseren Projekten hat sich folgende Richtlinie als Vorteil erwiesen:

Logger kennzeichnen Bereiche einer Applikation, der Log-Meldungen zugeordnet werden. Ein Bereich einer Applikation ist z.B. ein Baustein, Methode oder Namensraum (bei Libraries) die mit einem hierarchisch aufgebauten Namen gekennzeichnet werden. Die einzelnen Teile eines solchen Namens werden durch einen Punkt "." getrennt.

Beispiele:

- Eine Methode eines Libraries-Bausteins: `MyLib.PR_G_Foo.Init` (`MyLib`=Librarynamen, `PR_G_Foo`=Bautein, `Init`=Methode)
- Ein Baustein in einem Subsystem: `Communication.FB_Send` (`Communication`=Subsystem, `FB_Send`=Baustein)

Durch die hierarchische Gliederung können Log-Meldungen auch bei grossen Systemen mit Hunderten oder Tausenden Log-Meldungen schnell zugeordnet werden. Ebenso können z.B. auch einfach Meldungen eines Subsystems ausgefiltert oder in eine eigene Log-Datei umgeleitet werden.

Benutzung von Loggern

Alle Funktionen, die *Logger* unterstützten haben ein `L` im Namen:

- `F_Log` wird zu `F_LogL`
- `F_LogA1` wird zu `F_LogLA1`
- usw.

Diese Funktionen haben an 2. Stelle eine zusätzlichen Inputparameter vom Typ `T_MaxString` für den *Logger*. Wenn mehrere Log-Meldungen im gleichen Baustein ausgegeben werden, lohnt es sich eine Konstante dafür zu definieren. Der geänderte Code aus dem letzten Schritt sind mit *Logger* jetzt so aus:


```

PROGRAM MAIN
VAR CONSTANT
    sLogger          : STRING := 'MAIN';
END_VAR
VAR
    nCounter          : UINT;
    fbCountTime       : TON := (PT:=T#1S);
END_VAR
-----
IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN
    F_LogL(E_LogLevel.eInfo, sLogger, 'SPS Task gestartet.');
```

END_IF

```

fbCountTime(IN:=NOT fbCountTime.Q);
IF fbCountTime.Q THEN
    nCounter := nCounter + 1;
    F_LogLA1(E_LogLevel.eDebug, sLogger, 'Zähler geändert, neuer Wert {0}',
nCounter);
END_IF

PRG_TaskLog.Call();
```

Der Code befindet sich im Beispielprojekt unter den Namen "C_LogWithLogger".

Ausgabe im Log

Nachdem der geänderte Baustein geladen wurde, wird im Log-File in der dritten Spalte der *Logger* ausgegeben:

```

C:\ProgramData\log4TC\log\log4tc.log - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?
log4tc.log
1 2020-04-28 16:47:17.3320|INFO|MAIN|SPS Task gestartet.|[]
2 2020-04-28 16:47:18.3320|DEBUG|MAIN|Zähler geändert, neuer Wert 1|[]
3 2020-04-28 16:47:19.3520|DEBUG|MAIN|Zähler geändert, neuer Wert 2|[]
4
Normal text file length: 203 lines: 4 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

Was passiert wenn kein 'Logger' benutzt wird?

Logger sind ein integrierter Bestandteil von log4TC, aus diesen Grund wird intern immer ein *Logger* benutzt, auch wenn keiner angegeben wird. Der Logger in solchen Fällen ist in `Const.sGlobalLogger` definiert und hat den Wert `'_GLOBAL_'`.

Nächster Schritt

Integration von Context-Eigenschaften

Für log4TC sind Log-Meldungen mehr als simple Strings, die in Textdateien geschrieben werden. Jede Log-Meldung besteht aus einer variablen Anzahl von zwingend und optionalen Eigenschaften. Context-Properties sind solche optionale Eigenschaften.

Context-Eigenschaften sind ein weiterführendes Thema und können beim ersten Kontakt mit log4TC übersprungen werden. Um aber vom Logging-System die maximalen Nutzen ziehen zu können, lohnt sich aber die Einarbeitung.

Zweck der Context-Eigenschaften

Die Context-Eigenschaften einer Log-Meldungen ermöglichen es direkt und indirekt zusätzliche Daten einer Log-Message mitzugeben, zu verarbeiten, zu filtern und auszugeben. Der Context ist sehr ähnlich zu Argumenten einer Meldung, mit dem Unterschied, dass sie nicht direkt in der Log-Meldung erscheinen müssen.

Der Context existiert auf vier Ebenen:

- Task
- Verschachtelter Context (Nested Context)
- (Logger) - momentan noch nicht implementiert
- Log-Message

In dieser Einführung wird nur der letzte Typ beschrieben.

Context-Properties für Log-Messages

Um einer Log-Message einen Context mitzugeben, muss hierfür eine neue Variante der **F_Log*** Funktion verwendet werden:

```
F_LogLA1C(  
    E_LogLevel.eDebug,  
    sLogger,  
    'Zähler geändert, neuer Wert {0}',  
    nCounter,  
    F_LogContext().AddInt('MachineNo', 42)  
);
```

In diesen Beispiel wird an die von den vorherigen Schritten bereits vorhandene Log-Meldung eine Context-Eigenschaft mit dem Namen *MachineNo* und den Wert 42 hinzugefügt.

Der komplette Code sieht wie folgt aus:

```
PROGRAM MAIN
VAR CONSTANT
    sLogger          : STRING := 'MAIN';
END_VAR
VAR
    nCounter          : UINT;
    fbCountTime       : TON := (PT:=T#1S);
END_VAR

-----
IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN
    F_LogL(E_LogLevel.eInfo, sLogger, 'SPS Task gestartet. ');
END_IF

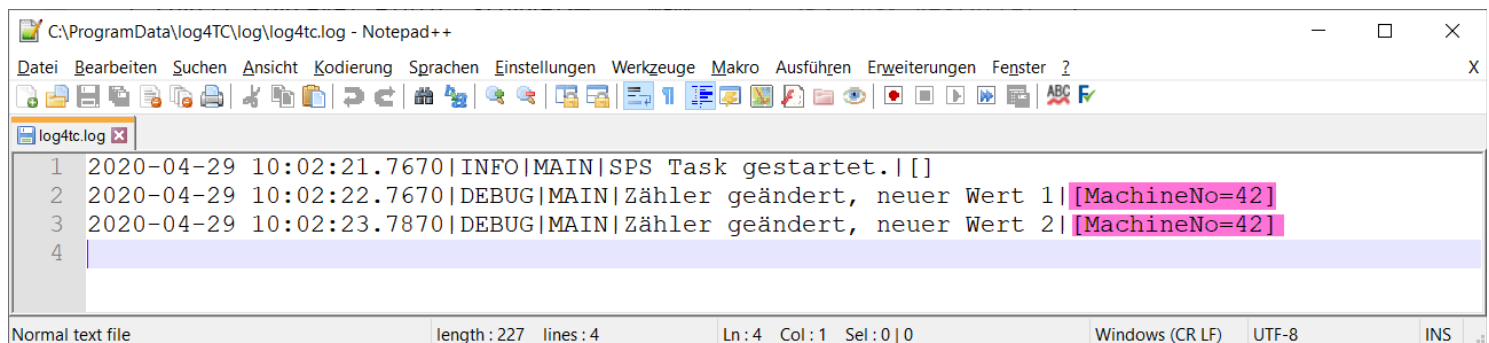
fbCountTime(IN:=NOT fbCountTime.Q);
IF fbCountTime.Q THEN
    nCounter := nCounter + 1;
    F_LogLA1C(
        E_LogLevel.eDebug,
        sLogger,
        'Zähler geändert, neuer Wert {0}',
        nCounter,
        F_LogContext().AddInt('MachineNo', 42)
    );
END_IF

PRG_TaskLog.Call();
```

Der Code befindet sich im Beispielprojekt unter den Namen "D_LogWithContext".

Log-Message

In der mitgelieferten Konfigurationsdatei werden die Context-Eigenschaften am Ende der Log-Meldung hinzugefügt:



```
C:\ProgramData\log4TC\log\log4tc.log - Notepad++
Datei Bearbeiten Suchen Ansicht Kodierung Sprachen Einstellungen Werkzeuge Makro Ausführen Erweiterungen Fenster ?
log4tc.log x
1 2020-04-29 10:02:21.7670|INFO|MAIN|SPS Task gestartet.|[]
2 2020-04-29 10:02:22.7670|DEBUG|MAIN|Zähler geändert, neuer Wert 1|[MachineNo=42]
3 2020-04-29 10:02:23.7870|DEBUG|MAIN|Zähler geändert, neuer Wert 2|[MachineNo=42]
4
Normal text file length: 227 lines: 4 Ln: 4 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

Nächster Schritt

Nächster Schritt

[Log-Meldungen mit Log4View beobachten](#)

Log-Meldungen mit Log4View beobachten

Für die Ausgabe, Speicherung und Weiterverarbeitung existieren eine fast endlose Anzahl an Werkzeugen. Ein grosser Vorteil von log4TC ist, dass es sich in etablierte bestehende Systeme integriert.

Ein Werkzeug für die Analyse und Anzeige von Log-Meldungen ist das Produkt [Log4View](#) von PROSA. Die Anwendung kann mit reduziertem Funktionsumfang kostenlos benutzt werden.

Installation von Log4View

Die Anwendung kann [Hier](#) heruntergeladen werden. Sie wird wie eine normale Windows Anwendung installiert.

Konfiguration der Log-Ausgabe in NLog

Log4View kann verschiedene Eingabeformate verarbeiten, aber das Log4J-XML hat sich bisher als sehr geeignet herausgestellt. Die NLog-Konfiguration in der log4TC Auslieferung enthält bereits alles Notwendige, damit solche Dateien geschrieben werden. Die neue Ausgabe ist im XML-Format und befindet sich im gleichen Ordner wie die bisher benutzte Log-Datei. Sie hat den Namen `log4tc.xml`. Der Inhalt ist XML, kann also prinzipiell in einem Texteditor geöffnet werden, ist aber schwieriger zu lesen.

Öffnen der `log4tc.xml` in LogView

Nach dem Starten von Log4View, kann mit *Start/Öffnen* die Log-Datei ausgewählt werden. Am einfachsten geht das, wenn man im Öffnen-Dialog im Pfad folgenden Text hineinkopiert und Enter betätigt: `%ProgramData%\log4TC\log\`. Danach wird die Datei `log4tc.xml` ausgewählt und mit Klick auf *Öffnen* bestätigt.

Jetzt muss noch das Format der Datei eingestellt werden. Die Einstellungen sollten den nachfolgenden Bildschirmfoto entsprechen:

Datei Empfänger hinzufügen



Einstellungen

Optionen

Empfangs-Filter



Dateiname

C:\ProgramData\log4TC\log\log4tc.xml



☒ Bisher geschriebene Meldungen lesen

☒ Ältere rollierte Dateien lesen (*.1, *.2, ...)

Dateinamen-Muster:

log4tc.xml.*

Passende Log-Dateien:

2

Rollierte Dateien begrenzen

☒ Nein

Rollierte Dateien zeigen

☐ Anzahl:

0

☐ Zeitraum

d, hh:mm

0, 00:03

☐ Zeitbereich

Lesen von

29.04.2020 10:16

Lesen bis

29.04.2020 10:19

Log Format

☒ XML

☐ Pattern

☐ JSON

Log Framework

☐ Log4net

☒ NLog

☒ Dialog immer anzeigen

☐ Einstellungen für xml Dateien merken

Tab

Neuer Tab



OK

Abbruch



Datei Empfänger

Bearbeiten

Einstellungen Optionen Empfangs-Filter ▶

Name
log4tc.xml

Farbe
☐ #FFEBF5FF

Kodierung
Unicode (UTF-8)

Puffergröße
500000 Meldungen

☐ Zeitzone für Receiver erzwingen

Zeitzone
(UTC+01:00) Amsterdam, Berlin, Bern, Rom, Stockholm, Wien

Zeitversatz
0 ms

OK Abbruch

Datei Empfänger

Nach dem Bestätigen mit *OK* wird die Datei geladen und im Fenster angezeigt. Man findet hier alle Informationen, die auch im Log-File waren, aber in strukturierter Form. So kann man z.B. zu einem Zeitstempel springen, Logger ausblenden, nur bestimmte Level anzeigen, usw. Der Nutzen erschliesst sich bei Log-Dateien mit mehreren Tausend Meldungen sehr schnell.

Log4View *

Datei Start Ansicht Anzeigeprofile Einstellungen Hilfe

Neu Bearbeiten Öffnen Datei Datenbank Clear All Message Views Bildlauf fortsetzen Bildlauf unterbrechen Alles auswählen Logger ausblenden Markierung an/aus An/Aus Zurück Weiter Alle löschen

Konfiguration Empfänger hinzufügen Meldungen Current Message View Auswahl Nur diesen Logger anzeigen Im Baum finden Alle Anzeigen Lesezeichen

Logger log4tc.xml

Suche

Filter An Bes... Em... Typ

Level	Datum	Logger	Meldung	Zeit
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2606 [MachineNo=42]	10:46:39.867
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2607 [MachineNo=42]	10:46:40.887
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2608 [MachineNo=42]	10:46:41.907
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2609 [MachineNo=42]	10:46:42.927
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2610 [MachineNo=42]	10:46:43.947
DEBUG	29.04.2020	MAIN	Zähler geändert, neuer Wert 2611 [MachineNo=42]	10:46:44.967
INFO	29.04.2020	MAIN	SPS Task gestartet. []	10:46:58.667
INFO	29.04.2020	MAIN	SPS Task gestartet. []	11:36:48.446

3094 Meldungen gepuffert 3094 Meldungen angezeigt Zeitraum: 29.04.2020 09:54 - 29.04.2020 12:13 Zoom Faktor: 100 %

Meldungsdetails

Meldung: Temperatur 29 [temperature=29, csv=True]

Quelle: log4tc.xml Prozess ID: 0 Thread: 1 Logger: MAIN

Exception:

Chart Meldungsdetails

Suche

3094 Meldungen empfangen 3094 Meldungen gepuffert 3094 Meldungen angezeigt 2 Logger 1 Empfänger 0 Lesefehler

Nächster Schritt

Protokollierung von strukturierten Werten

Protokollierung von strukturierten Werte

log4TC unterstützt das Prinzip von strukturierten Logging (Siehe auch [Message Templates](#)). Kern des Konzepts ist es alle Logging-Daten nicht als String zu übertragen, sondern alle Einzelteile in ihrer Ursprungsform zu übertragen und erst am Ende zusammenzusetzen. Der Vorteil ist, dass man während der Verarbeitung der Log-Meldung mehr Möglichkeiten hat. Ein Beispiel wird nachfolgend beschrieben, bei dem es darum geht, Temperaturen über log4TC in ein CSV-File zu schreiben.

Aufruf der Meldung mit strukturierter API

Im Beispiel steht in der Variable `fTemp` eine Temperatur, z.B. eines Schaltschranks. Der nachfolgende Code prüft, ob sich die Temperatur geändert hat und schreibt diese dann in das Log-File:

```
IF fTemp <> fPrevTemp THEN
    fPrevTemp := fTemp;

    F_LogLA1C(
        E_LogLevel.eInfo,
        sLogger,
        'Temperatur {temperature}',
        fTemp,
        F_LogContext().AddBool('csv', TRUE)
    );
END_IF
```

Dieser Aufruf hat zwei Neuerungen: Zum einen wird für das Temperatur-Argument in der Log-Meldung nicht der Platzhalter '{0}' verwendet sondern die strukturierte Form. Der Ausdruck `{temperature}` bedeutet das dieses Argument einen Namen, nämlich *temperature* zugeordnet wird. Zum anderen wird noch eine Context-Eigenschaft mit dem Namen `csv` auf den Wert `TRUE` gesetzt. Der Name und der Wert der Context-Eigenschaft wurde hier willkürlich gewählt, wichtig ist diese nur in Verbindung mit der NLog-Konfiguration.

Tip: In der Praxis wird der einfache Vergleich `fTemp <> fPrevTemp` sinnvollerweise durch einen Vergleich mit Hysterese ersetzt, da sonst selbst kleines rauschen beim Analogwandeln zu neuen Log-Meldungen führen.

Der komplette MAIN-Baustein sieht damit wie folgt aus:

```

PROGRAM MAIN
VAR CONSTANT

    sLogger          : STRING := 'MAIN';
END_VAR
VAR
    nCounter          : UINT;
    fbCountTime       : TON := (PT:=T#1S);
    fTemp             : REAL := 22.3;
    fPrevTemp         : REAL;
END_VAR
-----
IF _TaskInfo[GETCURTASKINDEXEX()].FirstCycle THEN
    // For Remote Log4TC Server change the AMS net ID and configure a route
    PRG_TaskLog.Init('127.0.0.1.1.1');
    F_LogL(E_LogLevel.eInfo, sLogger, 'SPS Task gestartet.');
```

END_IF

```

fbCountTime(IN:=NOT fbCountTime.Q);
IF fbCountTime.Q THEN
    nCounter := nCounter + 1;
    F_LogLA1C(
        E_LogLevel.eDebug,
        sLogger,
        'Zähler geändert, neuer Wert {0}',
        nCounter,
        F_LogContext().AddInt('MachineNo', 42)
    );
END_IF

IF fTemp <> fPrevTemp THEN
    fPrevTemp := fTemp;

    F_LogLA1C(
        E_LogLevel.eInfo,
        sLogger,
        'Temperatur {temperature}',
        fTemp,
        F_LogContext().AddBool('csv', TRUE)
    );
END_IF

PRG_TaskLog.Call();
```

Der Code befindet sich im Beispielprojekt unter den Namen "E_StructuredLogging".

Konfiguration von NLog

Das Ausgabe-Plugin *NLog* bietet eine Vielzahl an Funktionen an um Log-Meldungen auszugeben. In diesen Beispiel wird aber nur ein Ausschnitt betrachtet, der für das Verständnis notwendig ist.

Zunächst muss ein sog. **target** eingerichtet werden. Ein **target** ist die Konfiguration einer Ausgabe:

```
<target name="csvLogFile"
  xsi:type="File"
  fileName="${logdir}/log4tc.csv"
  <!-- weitere Optionen -->
  <layout xsi:type="CsvLayout"
    withHeader="true"
    delimiter="Tab">
    <column name="time" layout="${longdate}" />
    <column name="temp" layout="${event-
properties:item=temperature}" />
  </layout>
</target>
```

Im Beispiel wird eine Ausgabe in eine Datei (**xsi:type="File"**) konfiguriert, die ein CSV-Layout (**xsi:type="CsvLayout"**) benutzt. Danach werden zwei Spalten konfiguriert, eine für den Zeitstempel und eine für die Temperatur. Da die Temperatur als strukturiertes Element übergeben wird, kann direkt darauf mit **\${event-properties:item=temperature}** zugegriffen werden.

Als Nächstes dürfen an dieses **target** nur Log-Meldungen weitergeleitet werden, die auch relevant für das CSV sind. Eine Möglichkeit wäre zu prüfen, ob eine Meldung das **temperature** Argument besitzt oder nicht. Um einen anderen Weg zu zeigen, wird im Beispiel aber eine Context-Eigenschaft geprüft:

```
<logger name="*" minlevel="Info" writeTo="csvLogFile">
  <filter defaultAction="Ignore">
    <when condition="${event-properties:item=csv}" action="Log" />
  </filter>
</logger>
```

1. Zunächst werden alle Log-Meldungen Selektiert, die min. den Level "Info" haben (**minlevel="Info"**).
2. Für diese Meldungen wird geprüft, ob eine Context-Eigenschaft mit dem Namen **csv** vorhanden ist und wenn ja wird der Rückgabe-Wert ausgewertet (**condition="\${event-**

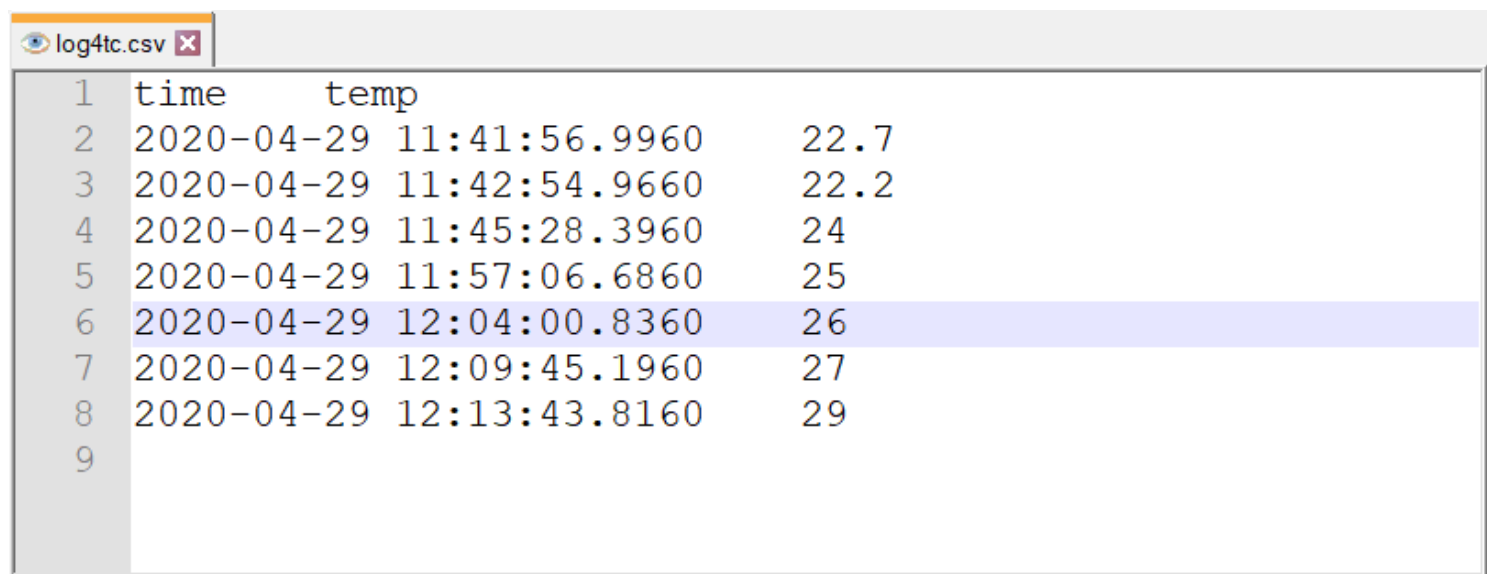
`properties:item=csv}"))`). Ist dieser `true` dann wird die Meldung an die CSV-Ausgabe weitergeleitet (`action="Log` in Verbindung mit `writeTo="csvLogFile"`).

3. Trifft die Bedingung nicht zu, dann wird sie für diesen `Target` ignoriert (`defaultAction="Ignore"`).

Tip: Das Ausgabe-Plugin *NLog* von log4TC ist eines der wichtigsten Ausgaben, da es seinerseits wiederum mit einer grossen Anzahl an *Targets* konfiguriert werden kann ([NLog Targets](#)). Es lohnt sich daher sich mit der Konfiguration von NLog vertraut zu machen.

CSV-Ausgabe

Lädt man das Programm und ändert man die Temperatur in der Variable `fTemp` einige Male von Hand, wird eine neue Log-Datei im CSV-Format angelegt mit folgenden Inhalt:



	time	temp
1		
2	2020-04-29 11:41:56.9960	22.7
3	2020-04-29 11:42:54.9660	22.2
4	2020-04-29 11:45:28.3960	24
5	2020-04-29 11:57:06.6860	25
6	2020-04-29 12:04:00.8360	26
7	2020-04-29 12:09:45.1960	27
8	2020-04-29 12:13:43.8160	29
9		

Wichtig ist, dass diese Meldung sowohl im CSV als auch im normalen Log ausgegeben werden, aber unterschiedlich formatiert.