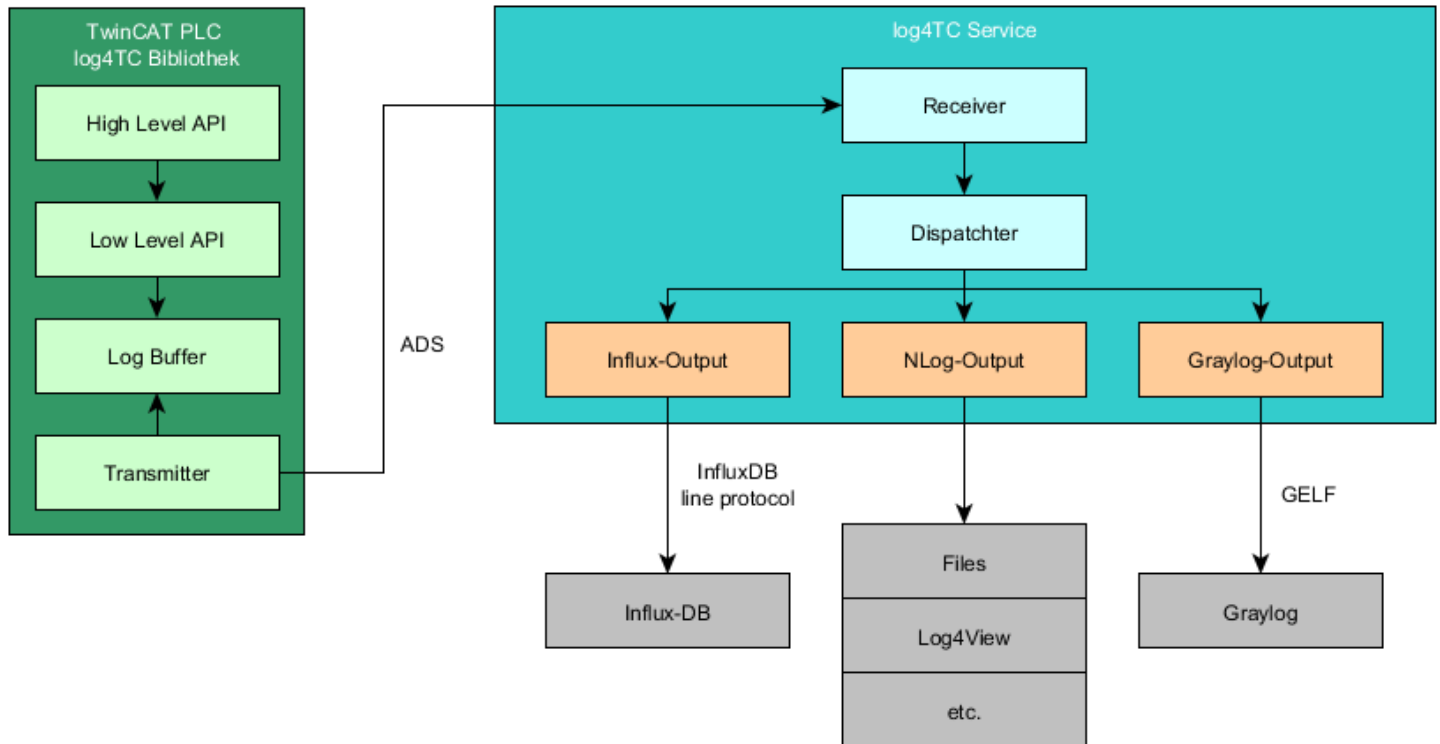


# mbc log4TC

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.

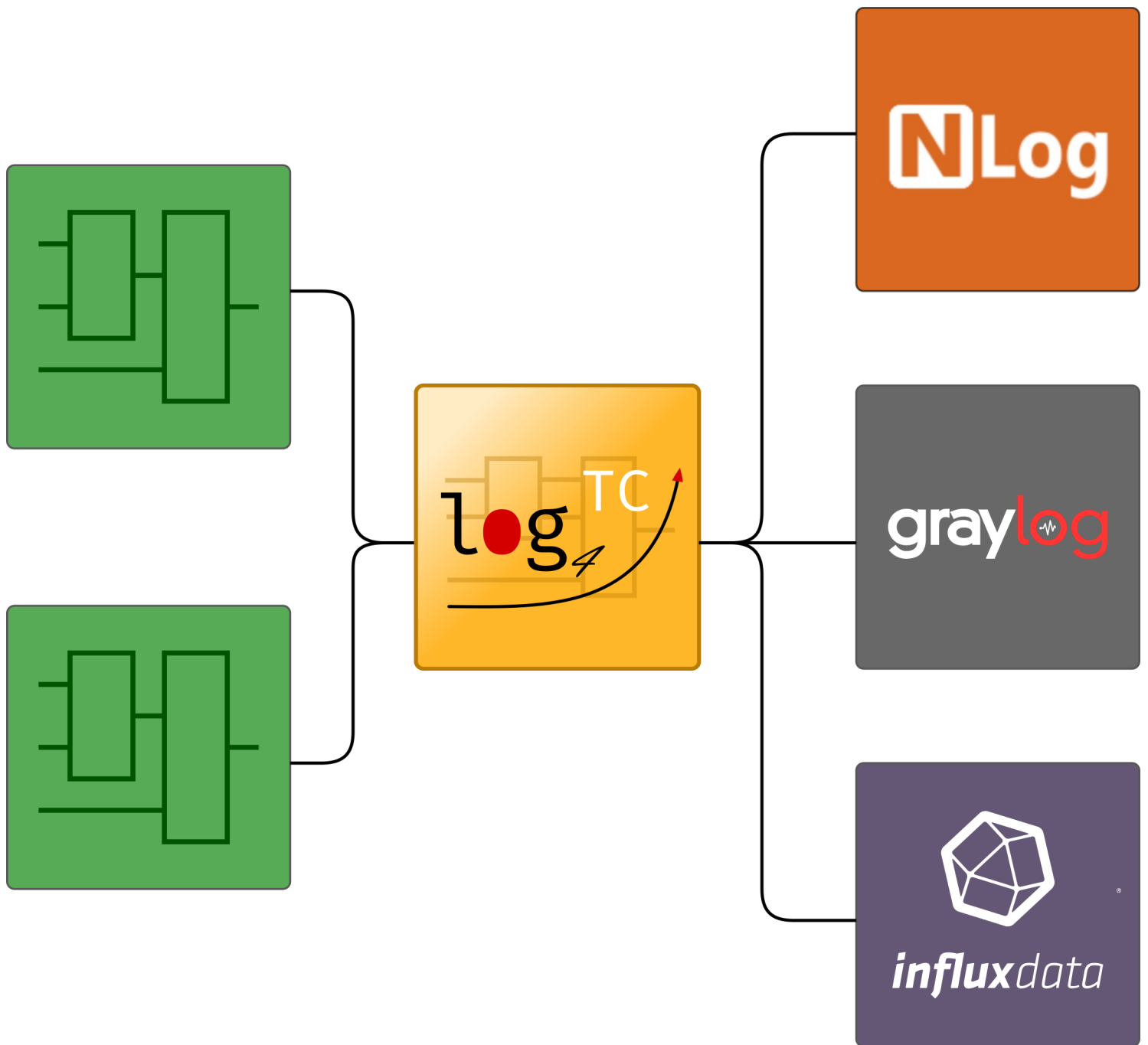


Der log4TC-Service wird normalerweise auf dem Rechner installiert, auf dem auch die SPS läuft, kann aber für bestimmte Einsatzzwecke auch auf einem anderen Rechner für mehrere Steuerung installiert werden.

## 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](#)

# Table of Contents

Werkzeuge .....	5
Formatierung .....	7
Loglevel .....	9
NLog-Ausgabeplugin .....	11
Graylog-Ausgabeplugin .....	17
InfluxDB-Ausgabeplugin .....	19
SQL-Ausgabeplugin .....	22
Installation .....	31
log4TC Servicekonfiguration .....	35

# Werkzeuge rund um log4TC

## Anzeigen von Log-Meldungen

### Log4View

Die Anwendung [Log4View](#) von Prosa ist eine unserer Empfehlungen für die Anzeige von mittleren oder grossen Anzahl an Log-Meldungen. Selbst die kostenlose Variante ist für viele Zwecke ausreichend.

Wichtig bei der Benutzung von Log4View mit log4TC sind folgende Punkte:

### NLog Konfiguration

NLog muss für die log4j-XML Ausgabe konfiguriert werden. NLog kommt mit einem Standard-Layout `log4jxmlevent`, das prinzipiell diese Anforderung erfüllt. log4TC stellt aber eine erweiterte Variante `mbclog4jxmlevent` mit folgenden Verbesserungen zur Verfügung:

- Thread-Id enthält die SPS Task-ID
- Zusätzliches Properties um die Message zu formatieren

Eine Beispiel-Konfiguration könnte so aussehen:

```
<extensions>
  <add assembly="Mbc.Log4Tc.Output.NLog"/>
</extensions>

<targets>
  <target name="xmlLogFile"
    xsi:type="File"
    encoding="utf-8"
    fileName="${logdir}/log4tc.xml"
    <!-- evtl. weiter File Optionen -->
```

```
layout="${mbclog4jxmlevent:includeAllProperties=true:message=${message} [{mbc-all-event-
properties}]]}">
  </target>
```

### Logging

[Log4View](#) bietet sehr gute Selektionsmöglichkeiten auf dem Logger und dem Level. Es lohnt sicher daher diese beiden Konsequenz im Code einzusetzen.

Leider bietet [Log4View](#) bis jetzt noch keine Unterstützung von Context-Properties und/oder structured Logging.

# Notepad++

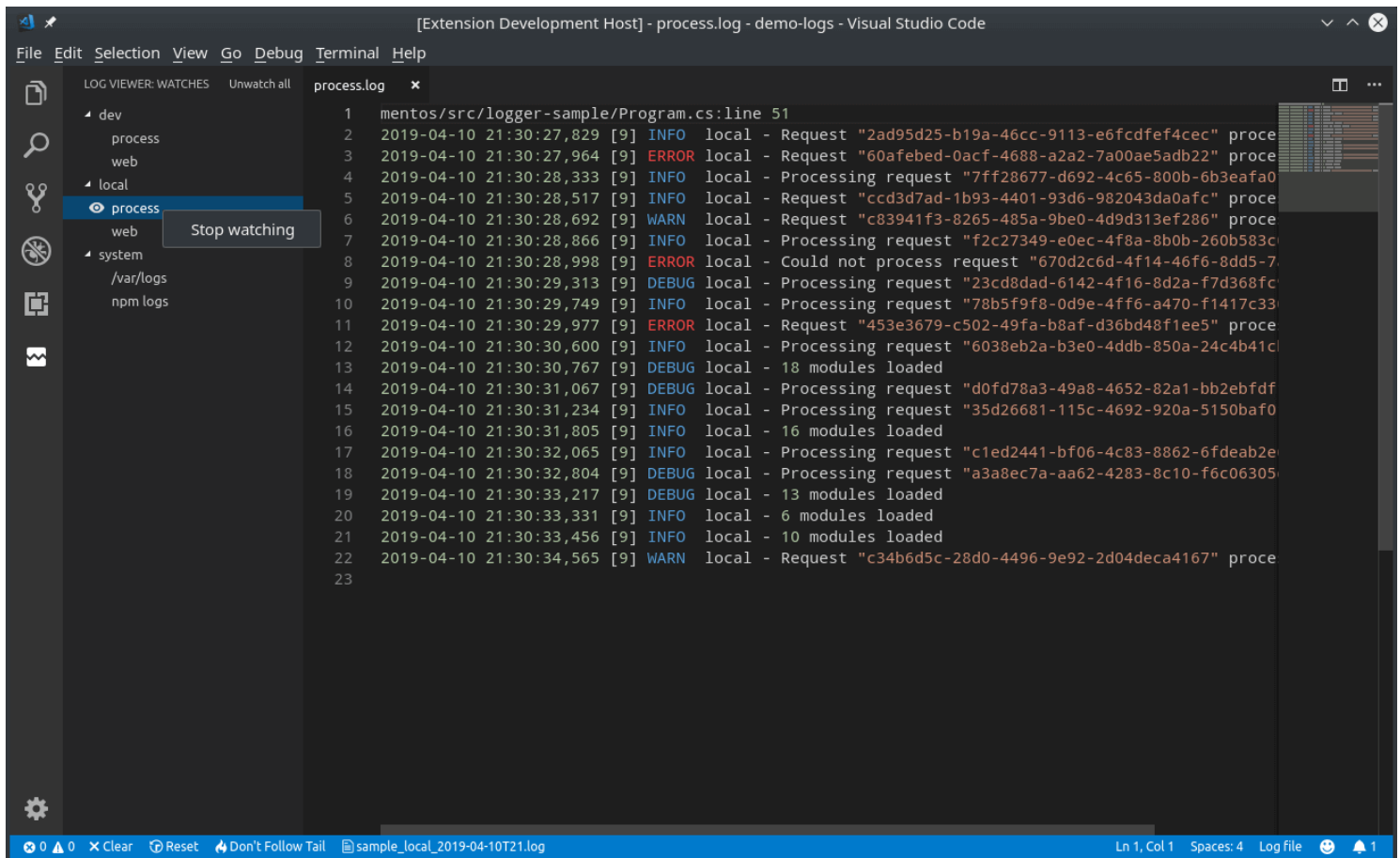
[Notepad++](#) bietet zwar keinen speziellen Modus für log4TC besitzt aber zwei Eigenschaften, die es interessant machen für einfache Logging Aufgaben:

- Im Menü "Ansicht" -> "Überwachen (tail -f)" kann Notepad++ angewiesen werden die Log-Datei laufend zu überwachen und bei Änderung automatisch einzulesen.
- Im Menü "Sprachen" kann nach eigenen Anforderungen ein Stil definiert werden, wie die einzelnen Blöcke in einem Log-File formatiert werden sollen

## Visual Studio Code

[Visual Studio Code](#) ist ähnlich wie Notepad++ eine universelle Plattform für Textverarbeitung. VS-Code bietet eine grosse Anzahl an Erweiterungen so z.B. auch für Log-File.

Eines davon *Log Viewer* bietet einfache Möglichkeiten für Log-Files:



# Format Strings für Meldungstexte

log4TC unterstützt mehrer Arten von Platzhaltern in Meldungstexten, entweder Positionsargumente oder benannte Argumente. Es gelten folgende Regeln:

- Argumten werden zwischen { und } Klammern geschrieben
- Die Klammern selbst können durch Verdopplung geschützt werden; {{ wird zu {
- Wenn alle Argumente nummeriert sind, wie z.B. {0}, {1} usw., dann werden die Argumentwerte gemäss der Nummer zugeordnet, also {0} verwendet den 1. Wert usw.
- Ist mindestens ein benanntest Argumente dabei wie z.B. {index} dann werden die Argumentwerte gemäss der Reihenfolge wie sie im Text vorkommen zugeordnet.
- Argumente können noch mit Optionen formatiert werden

Beispiel:

```
'Die Verarbeitung von {0} wurde {1} abgeschlossen'  
'Die Verarbeitung von {typ} wurde {status} abgeschlossen'
```

## Optionen für Argumente

Sowohl für positions als auch für benannte Argumente können noch weiter Optionen angegeben werden. Details dazu finden sich in [Composite Format String](#), nachfolgend aber eine Übersicht mit den wichtigsten Optionen.

Der grundlegende Aufbau ist wie folgt:

```
{Index[,Ausrichtung][:Format]}
```

bzw.

```
{Name[,Ausrichtung][:Format]}
```

*Index* oder *Name* ist entweder die Argument-Nr (0-basiert) oder der Argument Name. Beispiel: Von {0} bis {1}; Von {startTime} bis {endTime}.

Die Option *Ausrichtung* ist eine Ganzzahl mit Vorzeichen, welche die bevorzugte Grösse in Zeichen für die Ausgabe angibt. Positive Werte führen zu einer rechtsbündigen Ausrichtung, negative Werte zu einer linksbündigen. Beispiel:

- '({0,5})' {0}=42 => '( 42)'
- '({0,-5})' {0}=42 => '(42 )'

Die Option *Format* bestimmt wie der Typ des Arguments formatiert wird.

Beispiel für Zahlen:

- Decimal: `{0:D4}` ' {0}=42 => '(0042)' Nur für Ganzzahltypen. Parameter: Minimum Anzahl Ziffern.
- Exponential: `{0:E2}` ' {0}=42 => '(4.20E+001)' Parameter: Anzahl Nachkommastellen
- Fixed-point: `{0:F2}` ' {0}=42 => '(42.00)' Parameter: Anzahl Nachkommastellen
- Number: `{0:N2}` ' {0}=4200 => '(4'200.00)' Mit Gruppentrenner. Parameter: Anzahl Nachkommastellen
- Percent: `{0:P1}` ' {0}=0.42 => '(42.0%)' Parameter: Anzahl Nachkommastellen
- Hexadecimal: `{0:X4}` ' {0}=42 => '(002A)' Nur für Ganzzahltypen.



# Log-Level

Das log4Tc kennt sechs Log-Level:

- Trace
- Debug
- Info
- Warn
- Error
- Fatal

Die Log-Level sind geordnet, Trace hat die kleinste Ordnung, Fatal die grösste.

Grundsätzlich können die Bedeutung der Log-Level frei definiert werden, es ist aber empfehlenswert, wenn diese Projekt- oder Unternehmensweit definiert wird.

Log4Tc empfiehlt diese Richtlinien für die Benutzung von Log-Level:

## Level: Fatal

Der **Fatal**-Level sollte für Fehler verwendet werden, die verhindern, dass ein Programm komplett oder zum grossen Teil korrekt ausgeführt werden kann. Log-Meldungen von dieser Stufe bedeuten i.d.R. sofortiges Handeln und werden meist auch direkt weitergeleitet.

Beispiel: Der Safety-Controller konnte wegen eines HW-Fehlers nicht initialisiert werden.

## Level: Error

Der **Error**-Level kennzeichnet Meldungen, die von Fehlern stammen, die i.d.R. auch zu Problemen im Programm führen. Im Gegensatz zum **Fatal**-Level sind hier aber nur Teile der Software betroffen.

Auch diese Meldungen werden normalerweise direkt an einen Operator/Service weitergeleitet.

Beispiele: Kommunikation zu einem übergeordneten System ausgefallen.

## Level: Warn

Zustände, die noch nicht zu einem Fehler führen, aber bereits ein baldiges Eingreifen eines Users erfordern, können mit **Warn** gemeldet werden. Normalerweise läuft die Software noch problemlos weiter. Die Reaktion auf diese Meldungen ist häufig verzögert.

Beispiel: Zu wenig Kühlmittel im System.

## Level: Info

Der **Info**-Level soll wichtige Zustandsänderung der Software protokollieren. Häufig sind diese Informationen wichtig um Fehler besser zuordnen zu können.

Beispiel: Ein neues Rezept mit der ID 42 wurde geladen.

## Level: **Debug**

Dieser Level wird häufig von Entwicklern verwendet um weitere detaillierte Zustandsänderungen zu verfolgen. Im Normalfall sind diese Information nur für Entwickler notwendig.

Beispiel: Der Aufruf des Sendebaustein hat 3.2s benötigt.

## Level: **Trace**

Der **Trace**-Level wird ebenfalls von Entwickler verwendet um weitergehende interne Zustände zu verfolgen. Entwickler verwenden diesen Level für die Analyse von bestehenden Problemen. Log-Messages dieses Levels werden häufig nicht dauerhaft abgespeichert.

# NLog Ausgabe-Plugin

Log4TC wird standardmässig mit dem NLog Ausgabe-Plugin ausgeliefert, an dem alle Log-Meldungen von log4TC weitergereicht werden. Die Detailkonfiguration kann in [NLog-Project](#) nachgeschlagen werden.

## Konfiguration in log4TC

log4TC erwartet die Konfiguration von NLog in %ProgramData%\log4TC\config\NLog.config, am einfachsten kommt man in den Ordner über den Link im Startmenü, der beim Installieren von log4TC angelegt wird.

Die bei der Installation mit ausgelieferte Konfiguration sind auf die [Einführung](#) ausgelegt und sollte daher für eigene Projekt angepasst werden. Nachfolgende werden zwei Konfiguration vorgestellt, die als Basis für eigene Anwendungen verwendet werden können.

## Einfaches Text-Logging

Diese Konfiguration schreibt die Log-Meldungen von log4TC in normale Text-Files, die von jeden Editor gelesen werden können (siehe auch [Tools](#)).

```
<?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"
  throwConfigExceptions="true"
  autoReload="true"
  internalLogLevel="Info"
  throwExceptions="true">

  <!--
    See https://github.com/nlog/nlog/wiki/Configuration-file
    for information on customizing logging rules and outputs.

    See also for targets: https://nlog-project.org/config/?tab=targets
    See also for placeholders: https://nlog-project.org/config/?tab=layout-renderers
  -->

  <extensions>
    <add assembly="Mbc.Log4Tc.Output.NLog"/>
  </extensions>

  <variable name="logdir" value="{specialfolder:folder=CommonApplicationData}\log4TC\log"/>

  <targets>
```

```

<target name="textLogFile"
  xsi:type="File"
  createDirs="true"
  encoding="utf-8"
  archiveFileName="${logdir}/log4tc.log.{#}"
  fileName="${logdir}/log4tc.log"
  maxArchiveFiles="5"
  archiveAboveSize="10485760"
  archiveNumbering="Rolling"
  layout="${longdate}|${level:uppercase=true}|${logger}|${message}|[${mbc-all-
event-properties}]">
  </target>
</targets>

<rules>
  <!--Levels: Trace, Debug, Info, Warn, Error, Fatal, Off-->
  <logger name="*" minlevel="Debug" writeTo="textLogFile" />
</rules>
</nlog>

```

Die Konfiguration hat folgende Eigenschaften:

- Die Ausgabe erfolgt in %ProgramData%\log4TC\log\log4Tc.log. Der Pfad kann über die Variable `logdir` geändert werden.
- Das Log-File wird max. 10 MByte gross, danach wird es archiviert. Es werden max. 5 Archive aufbewahrt, bevor endgültig gelöscht wird.
- Das Ausgabeformat ist: <PLC-Zeitstempel>|<Level>|<Logger>|<Log-Message>| [<Context-Attribute>].
- Es werden alle Meldungen ab Level `Debug` und höher geloggt.

## Ausgabe für Log4View

Diese Konfiguration schreibt die Log-Meldungen in ein XML-Format, dass von [Log4View](#) gelesen werden kann.

```

<?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"
  throwConfigExceptions="true"
  autoReload="true"
  internalLogLevel="Info"
  throwExceptions="true">

  <!--

```

See <https://github.com/nlog/nlog/wiki/Configuration-file>  
for information on customizing logging rules and outputs.

See also for targets: <https://nlog-project.org/config/?tab=targets>

See also for placeholders: <https://nlog-project.org/config/?tab=layout-renderers>

-->

```
<extensions>
  <add assembly="Mbc.Log4Tc.Output.NLog"/>
</extensions>

<variable name="logdir" value="${specialfolder:folder=CommonApplicationData}\log4TC\log"/>

<targets>
  <target xsi:type="File"
    name="xmlLogFile"
    createDirs="true"
    encoding="utf-8"
    archiveFileName="${logdir}/log4tc.xml.{#}"
    fileName="${logdir}/log4tc.xml"
    maxArchiveFiles="5"
    archiveAboveSize="10485760"
    archiveNumbering="Rolling"
    layout="${mbclog4jxmlevent:includeAllProperties=true:message=${message} [{${mbc-
all-event-properties}}]}">
  </target>
</targets>

<rules>
  <!--Levels: Trace, Debug, Info, Warn, Error, Fatal, Off-->
  <logger name="*" minlevel="Debug" writeTo="xmlLogFile" />
</rules>
</nlog>
```

Die Konfiguration hat folgende Eigenschaften:

- Die Ausgabe erfolgt in %ProgramData%\log4TC\log\log4Tc.xml. Der Pfad kann über die Variable `logdir` geändert werden.
- Das Log-File wird max. 10 MByte gross, danach wird es archiviert. Es werden max. 5 Archive aufbewahrt, bevor endgültig gelöscht wird.
- Im Meldungstext werden noch zusätzlich alle Context-Properties eingefügt, sofern welche vorhanden sind.
- Es werden alle Meldungen ab Level `Debug` und höher geloggt.

## Ausgabe für Azure ApplicationInsight

Damit NLog die Ausgabe für [Azure ApplicationInsight](#) unterstützt, muss ein `ApplicationInsightsTargetLog4Tc` target wie folgt konfiguriert werden.

```
<?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"
  throwConfigExceptions="true"
  autoReload="true"
  internalLogLevel="Info"
  throwExceptions="true">

  <!--
    See https://github.com/nlog/nlog/wiki/Configuration-file
    for information on customizing logging rules and outputs.

    See also for targets: https://nlog-project.org/config/?tab=targets
    See also for placeholders: https://nlog-project.org/config/?tab=layout-renderers
    -->

  <extensions>
    <add assembly="Mbc.Log4Tc.Output.NLog"/>
  </extensions>

  <targets>
    <target xsi:type="ApplicationInsightsTargetLog4Tc" name="appi">
      <instrumentationKey>[YourAppiInstrementationKey]</instrumentationKey>
      <!-- Can be repeated with more Custom Properties -->
      <contextproperty name="instance" layout="plc1" />
    </target>
  </targets>

  <rules>
    <!--Levels: Trace, Debug, Info, Warn, Error, Fatal, Off-->
    <logger name="*" minlevel="Trace" writeTo="appi" />
  </rules>
</nlog>
```

## NLog-Erweiterungen

Log4TC liefert einige Erweiterungen für NLog mit.

### Layout `mbclog4jxmlevent` (Erweiterung für `log4jxmlevent`)

Dieses Layout erweitert bzw. passt das in das NLog integrierte `log4xmlevent` um folgende Eigenschaften an:

- Alle C#-spezifischen Einstellungen sind weggefallen (kein Mdc, Mdlc, Ndc, CallSite und SourceInfo).
- Die TwinCAT Task-ID wird als Thread-ID ausgegeben und kann in Log4View unter diesen Namen direkt abgelesen werden
- Das Feld `log4japp` wird mit der Quelle (z.B. `172.16.23.20.1.1:350`) aufgefüllt
- Für `AppInfo` und `Message` kann ein eigenes Layout definiert werden.

Beispiel:

```
layout="${mbclog4xmlevent:includeAllProperties=true:message=${message} [${mbc-all-event-properties}]]">
```

Properties:

- `IdentXml` (Boolean) - Gibt an, ob das XML formatiert ausgegeben werden soll (Default: `false`)
- `AppInfo` (Layout) - Inhalt des `log4japp`-Attributs (Default: Kombination aus `_TcAppName_` und `_TcProjectName_`)
- `Message` (Layout) - Die auszugebene Meldung (Default: Der formatierte Meldungstext)
- `LoggerName` (Layout) - Der auszugebene Logger (Default: Der Loggername)
- `IncludeAllProperties` (Boolean) - Wenn `true` werden alle (Context-)Eigenschaften mit ausgegeben.

## LayoutRenderer `mbc-all-event-properties` (Erweiterung für `all-event-properties`)

Dieser LayoutRender hat eine zusätzliche Option `ExcludeStandard` (Default: `true`), die verhindert, dass die Standard-Properties, die jede Meldung besitzt mit ausgegeben werden.

## log4TC Standard-Properties

Alle NLog-Meldungen bekommen unabhängig vom Context und den Argumenten folgenden Properties:

- `_TcTaskIdx_` - Der TwinCAT Task-Index (1-x)
- `_TcTaskName_` - Der Name der TwinCAT Task
- `_TcTaskCycleCounter_` - Der Wert des Task-Zyklusähler (alle Meldungen vom gleichen Zyklus haben den gleichen Wert)
- `_TcAppName_` - Der Name der TwinCAT Application
- `_TcProjectName_` - Der Name des TwinCAT Projekts
- `_TcOnlineChangeCount_` - Anzahl der Online-Changes
- `_TcLogSource_` - Die Quelle der Log-Meldung (AdsNetId mit AdsPort)
- `_TcHostname_` - Der Hostname des Rechners, vom dem die Meldung stammt

# Tipps und Rezepte

## Filtern mit Properties

Um Log-Meldungen mit Properties zu filtern (Argument und Context), kann der NLog-Filter verwendet werden:

```
<logger ...>
  <filter defaultAction="Ignore">
    <when condition="${event-properties:item=foobar}" action="Log" />
  </filter>
</logger>
```

Details dazu finden sich in [Filtering log message](#).



# Graylog Ausgabe-Plugin

Graylog ist ein Log-Managementsystem, das zentral Log-Meldungen entgegennimmt, speichert und zur Echtzeitanalyse bereitstellt. Graylog benutzt intern Elasticsearch und lässt sich horizontal skalieren, aber auch nur auf einem einzelnen Rechner installieren. Graylog kann als Open Source oder als Enterprise Version eingesetzt werden.

Log4TC wird standardmässig mit dem Graylog Ausgabe-Plugin ausgeliefert. Details zur Graylog können auf <https://docs.graylog.org/> nachgelesen werden.

## Konfiguration in log4TC

Um Meldungen an Graylog ausgeben zu können, muss der log4TC-Service zuerst konfiguriert werden. Eine einfache Konfiguration (%ProgramData%\log4TC\config\appsettings.json) sieht wie folgt aus:

```
{
  {
    "Logging": {
      "LogLevel": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    {
      "Outputs": [
        {
          "Type": "graylog",
          "Config": {
            "GraylogHostname": "localhost"
          }
        }
      ]
    }
  }
}
```

Die wesentliche Konfiguration der Graylogausgabe befindet sich in der **Outputs**-Liste.

```
{
  "Type": "graylog",
  "Config": {
    "GraylogHostname": "localhost", # Hostname oder IP, optional, Default "localhost"
    "GraylogPort": 12201,          # UDP-Port auf Graylog-Server, optional,
    Default 12201
  }
}
```

```
    "GelfCompression": "Gzip",      # GELF-Kompression, optional, Default gzip.
Alternativen: "None"
  }
}
```

Log4TC kommuniziert mit dem Graylog-Server über Gelf-UDP, daher muss auf dem Server ein entsprechender Input konfiguriert werden, was häufig schon der Fall ist.

Graylog kann nativ installiert werden, wir empfehlen aber den Betrieb über Docker, zumindest wenn Graylog abseits der TwinCAT-Runtime installiert werden kann (Docker läuft nicht zusammen mit der TwinCAT-Runtime bzw. umgekehrt).

### NOTE

Ein Beispiel Docker-Compose befindet sich auf Github im Graylog-Beispielordner [https://github.com/mbc-engineering/log4TC/blob/master/source/TwinCat\\_Examples/graylog/docker-compose.yml](https://github.com/mbc-engineering/log4TC/blob/master/source/TwinCat_Examples/graylog/docker-compose.yml) .

## log4TC Log-Meldungen in Graylog

Log4TC versucht bei der Ausgabe in Graylog alle strukturierten Elemente zu erhalten. Eine Graylog-Meldung hat folgende Felder:

- **appName** - Der Name der TwinCAT Application (z.B. **Port\_851**)
- **clockTimestamp** - Der Zeitstempel der Windowsuhr (geringe Genauigkeit)
- **fullMessage** - Die Log-Meldung ohne Variablenersetzung. Kann für Suche und Selektion verwendet werden um gleichartige Meldungen zu selektieren.
- **level** - Der Log-Level als Syslog-Wert. Graylog kann diesen Wert mit Decorators in Text umwandeln
- **logger** - Der Loggername der Log-Meldung.
- **message** - Die Log-Meldung mit Variablenersetzungen.
- **onlineChangeCount** - Anzahl der Online-Changes
- **projectName** - Der Name des SPS-Projekts.
- **source** - Die Ads-Net-Id von der die Log-Message empfangen wurde.
- **hostname** - Der Hostname von dem die Log-Message empfangen wurde.
- **taskCycleCounter** - Der Wert des Task-Zykluszähler (alle Meldungen vom gleichen Zyklus haben den gleichen Wert)
- **taskIndex** - Der TwinCAT Task-Index (1-x)
- **taskName** - Der Name der TwinCAT Application
- **timestamp** - Der PLC-Zeitstempel (hohe Genauigkeit)

# InfluxDB Ausgabe-Plugin

InfluxDB ist eine auf Zeitreihendaten spezialisierte Datenbank. Dies ermöglicht es zeitlich anfallende Daten (zyklisch und azyklisch) zu speichern und mit einer SQL-artigen Sprache abzufragen. InfluxDB ist OpenSource, eine Cloud-Lösung ist in Vorbereitung.

Zur Anzeige der Daten in InfluxDB wird eine zusätzliche Anwendung mitgeliefert, Chronograf. Hierbei handelt es sich um eine Web-Anwendung, mit dem Queries oder andere Anfragen abgesetzt werden und auch in Charts visualisiert werden können. Wir empfehlen aber für weitergehende Anwendung Grafana, das ein Influx-DB Input besitzt und zur Visualisierung mehr Möglichkeiten bietet.

Log4TC wird standardmässig mit dem InfluxDB Ausgabe-Plugin ausgeliefert. Details zu InfluxDB können auf <https://www.influxdata.com/> nachgelesen werden.

## Konfiguration in log4TC

Um Meldungen an Influx ausgeben zu können, muss der log4TC-Service zuerst konfiguriert werden. Eine einfache Konfiguration (%ProgramData%\log4TC\config\appsettings.json) sieht wie folgt aus:

```
{
  {
    "Logging": {
      "LogLevel": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    "Outputs": [
      {
        "Type": "influxdb",
        "Filter": { "Logger": "influx.*" },
        "Config": {
          "Url": "http://localhost:8086",
          "Database": "log4tc",
          "Format": "arguments"
        }
      }
    ]
  }
}
```

Die wesentliche Konfiguration von Influx befindet sich in der **Outputs**-Liste.

```

{
  "Type": "influxdb",
  "Filter": { "Logger": "influx.*" }, # optional zur Selektioni der Meldungen
  "Config": {
    "Url": "http://localhost:8086", # Adresse und Port, auf dem InfluxDB
Daten entgegenimmt
    "Username": "", # Benutzername, optional, Default kein Benutzer
    "Password": "", # Passwort, optional, Default kein Passwort
    "Database": "log4tc", # Datenbank, erforderlich
    "RetentionPolicy": "", # Retention-Policy, optional, Default Standard-
Retention des Servers
    "WriteBatchSize": 1, # Anzahl Meldungen, die zusammen geschrieben werden,
optional, Default 1
    "WriteFlushIntervalMillis": 1000, # Max. Wartezeit bevor ein nicht vollständiger Batch
geschrieben wird, optional, Default 1000
    "Format": "arguments", # Format, mit dem die Daten geschrieben werden,
optiona, Default "arguments", Alternativen: "syslog"
    "SyslogFacilityCode": 16 # Nummer der verwendeten Syslog-Facility bei
Format "syslog"
  }
}

```

InfluxDB kann nativ installiert werden, wir empfehlen aber den Betrieb über Docker, zumindest wenn Docker abseits der TwinCAT-Runtime installiert werden kann (Docker läuft nicht zusammen mit der TwinCAT-Runtime bzw. umgekehrt).

#### **NOTE**

Ein Beispiel Docker-Compose befindet sich auf Github im InfluxDB-Beispielordner [https://github.com/mbc-engineering/log4TC/blob/master/source/TwinCat\\_Examples/influx\\_with\\_message/docker-compose.yml](https://github.com/mbc-engineering/log4TC/blob/master/source/TwinCat_Examples/influx_with_message/docker-compose.yml).

## log4TC Log-Meldungen in InfluxDB

Log4TC kann Log-Meldungen in zwei Formaten in die InfluxDB schreiben.

### Format **syslog**

Bei diesen Format wird die komplette Log-Message im Syslog-Format geschrieben. In diesen Fall können die Meldungen mit dem in Chronograf integrierten LogViewer angezeigt, gefilter und gesucht werden.

## NOTE

Die Möglichkeiten zur Anzeige und Verarbeitung der Log-Messages ist relativ einfach und kann für viele Anwendungen ausreichen, vor allem wenn aus anderen Gründen Influx vorhanden ist. Für mehr Möglichkeiten zur Log-Message analyse empfehlen wir aber Graylog.

## Format arguments

Bei diesen Format werden nur die strukturierten Argumente einer Log-Message in die Influx-DB geschrieben. Dieses Format bietet sich an, wenn man die Daten in Charts anzeigen möchte.

Der Ablauf ist wie folgt:

- Sobald eine Meldung in der Ausgabe ankommt, wird geprüft ob sie strukturierte Argumente enthält (z.B. `Anfrage mit ErrorCode={code} abgeschlossen`). Wenn nicht, wird diese Meldung übersprungen.
- Es wird ein Measurement erzeugt mit dem Namen des Loggers.
- Der Zeitstempel entspricht dem PLC-Zeitstempel in us-Auflösung.
- Alle Context-Attribute der Log-Message werden als Tags hinzugefügt.
- Zusätzlich werden folgende Standard-Tags gesetzt:
  - `level` - Der Log-Level
  - `source` - Die Ads-Net-Id von der die Log-Message empfangen wurde.
  - `hostname` - Der Hostname von dem die Log-Message empfangen wurde.
  - `taskName` - Der Name der TwinCAT Application
  - `taskIndex` - Der TwinCAT Task-Index (1-x)
  - `appName` - Der Name der TwinCAT Application (z.B. `Port_851`)
  - `projectName` - Der Name des SPS-Projekts.
- Alle strukturierten Argumente werden mit ihren Namen und Wert als Felder zur Messung hinzugefügt. Im einleitenden Beispiel wäre das das Feld `code` mit dem Wert des Arguments.

Wir empfehlen alle Werte die zusammengehören in der gleichen Log-Meldung zu verschicken, da sie dann zusammen in die Messung geschrieben werden.

## NOTE

Soll eine Log-Meldung nur an Influx ausgegeben werden, kann die Meldung nur aus den Platzhaltern bestehen z.B. `{code}` anstatt `Anfrage mit ErrorCode={code} abgeschlossen`.

# SQL Ausgabe-Plugin

Das SQL-Ausgabe-Plugin schreibt log4TC-Meldungen in eine relationale Datenbank. Log4TC unterstützt derzeit vier Datenbanken: MySql/MariaDB, PostgreSql und MS SQLServer.

## Vorbereitung der Datenbank

Um Meldungen an eine SQL-Datenbank ausgeben zu können muss das Schema zuerst in der Datenbank vorbereitet werden. Log4TC unterstützt momentan zwei Varianten. Die DDL's zu den jeweiligen Datenbanken befinden sich am [Ende dieses Artikels](#).

## Konfiguration in log4TC

Die Konfiguration für den log4TC-Service (%ProgramData%\log4TC\config\appsettings.json) sieht wie folgt aus:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "Outputs": [
    {
      "Type": "sql",
      "Config": {
        "Driver": "MySql, Postgres oder SqlServer",
        "ConnectionString": "siehe Text",
        "Scheme": "SimpleFlat oder FullFlat"
      }
    }
  ]
}
```

### Driver

Definiert den Treiber für den Zugriff auf die Datenbank. Momentan wird log4TC mit Treibern für folgende Datenbank ausgeliefert:

- **MySql:** [MySql](#) oder [MariaDB](#)
- **Postgres:** [Postgres](#)
- **SqlServer:** [MS SQL-Server](#)

## ConnectionString

Legt die Verbindungseinstellungen zu der ausgewählten Datenbank fest. Der String ist Abhängig von der Datenbank. Die Webseite <https://www.connectionstrings.com/> liefert eine gute Übersicht über weitergehende Parameter.

- **MySQL/MariaDB:** `Server=dbserverhost;Port=3306;Database=databasename;Uid=user;Pwd=password;`
- **Postgres:** `User ID=user;Password=password;Host=dbserverhost;Port=5432;Database=databasename;Pooling=true;`
- **SqlServer:** `Server=dbserverhost,1433;Database=dbserverhost;User Id=user;Password=password;`

## Schema

Legt fest, in welchen Format die log4TC-Meldungen geschrieben werden sollen. Momentan werden zwei Varianten unterstützt:

- **SimpleFlat** - Einfaches Schema mit einer Tabelle; strukturierte Daten werden nicht geschrieben (nur im Message-Text)
- **FullFlat** - Schreibt die Context- und Argument-Strukturen für jede Meldung in separate Tabellen

## log4TC SQL-Schema

### NOTE

Benötigen Sie ein anderes Schema oder eine andere Datenbank? Kontaktieren Sie uns!

## SimpleFlat

Dies ist das einfachste Format und entspricht in etwas den Informationsgehalt von Log-Files.

Tabelle **log\_entry**:

Spalte	Bedeutung
id	Eindeutige ID (Primary-Key), wird von der Datenbank vergeben
source	Die Ads-Net-Id von der die Log-Message empfangen wurde.
hostname	Der Hostname von dem die Log-Message empfangen wurde.
formatted_message	Die formatierte Meldung, die in der SPS geschrieben wurde.
logger	Der Loggername der Log-Meldung.

Spalte	Bedeutung
level	Der Log-Level der Log-Meldung.
plc_timestamp	Der (interne) PLC-Zeitstempel der TwinCAT-Runtime, wenn die Meldung erzeugt wurde.
clock_timestamp	Der Zeitstempel der Windowsuhr (geringe Genauigkeit)
task_index	Der TwinCAT Task-Index (1-x)
task_name	Der Name der TwinCAT Application
task_cycle_counter	Der Wert des Task-Zykluszähler (alle Meldungen vom gleichen Zyklus haben den gleichen Wert)
app_name	Der Name der TwinCAT Application (z.B. <b>Port_851</b> )
project_name	Der Name des SPS-Projekts.
onlinechange_count	Anzahl der Online-Changes

## FullFlat

Dieses Format enthält alle log4TC-Daten inkl. Argumente und Context in einem flachen Format. Dieses Schema besteht aus drei Tabellen:

Tabelle **log\_entry**:

Spalte	Bedeutung
id	Eindeutige ID (Primary-Key), wird von der Datenbank vergeben
source	Die Ads-Net-Id von der die Log-Message empfangen wurde.
hostname	Der Hostname von dem die Log-Message empfangen wurde.
formatted_message	Die formatierte Meldung, die in der SPS geschrieben wurde.
message	Die rohe Meldung mit Platzhaltern, so wie sie in der SPS geschrieben wurde.
logger	Der Loggername der Log-Meldung.
level	Der Log-Level der Log-Meldung.



Spalte	Bedeutung
plc_timestamp	Der (interne) PLC-Zeitstempel der TwinCAT-Runtime, wenn die Meldung erzeugt wurde.
clock_timestamp	Der Zeitstempel der Windowsuhr (geringe Genauigkeit)
task_index	Der TwinCAT Task-Index (1-x)
task_name	Der Name der TwinCAT Application
task_cycle_counter	Der Wert des Task-Zykluszähler (alle Meldungen vom gleichen Zyklus haben den gleichen Wert)
app_name	Der Name der TwinCAT Application (z.B. <b>Port_851</b> )
project_name	Der Name des SPS-Projekts.
onlinechange_count	Anzahl der Online-Changes

Tabelle **log\_argument**:

Spalte	Bedeutung
id	Eindeutige ID (Primary-Key), wird von der Datenbank vergeben
log_entry_id	Referenz die Tabelle log_entry
idx	Der Argument-Index in der zugehörigen Log-Message
value	Der Argument-Wert als String
type	Der Ursprungstyp des Arguments

Tabelle **log\_context**:

Spalte	Bedeutung
id	Eindeutige ID (Primary-Key), wird von der Datenbank vergeben
log_entry_id	Referenz die Tabelle log_entry
name	Der Context-Name der zugehörigen Log-Message

Spalte	Bedeutung
value	Der Context-Wert als String
type	Der Ursprungstyp des Context-Attributts

# Anhang: DDL für Datenbank-Schemas

## MySql/MariaDB

### SimpleFlat

```
CREATE TABLE IF NOT EXISTS log_entry (
  id                BIGINT AUTO_INCREMENT PRIMARY KEY,
  source            VARCHAR(30) NOT NULL,
  hostname          VARCHAR(30) NOT NULL,
  formatted_message TINYTEXT NOT NULL,
  logger            VARCHAR(255) NOT NULL,
  level             ENUM('trace', 'debug', 'info', 'warn', 'error', 'fatal') NOT NULL,
  plc_timestamp     TIMESTAMP NOT NULL,
  clock_timestamp   TIMESTAMP NULL,
  task_index        TINYINT NOT NULL,
  task_name         VARCHAR(63) NOT NULL,
  task_cycle_counter INT NOT NULL,
  app_name          VARCHAR(63) NOT NULL,
  project_name      VARCHAR(63) NOT NULL,
  onlinechange_count INT NOT NULL
);
```

### FullFlat

```
CREATE TABLE IF NOT EXISTS log_entry (
  id                BIGINT AUTO_INCREMENT PRIMARY KEY,
  source            VARCHAR(30) NOT NULL,
  hostname          VARCHAR(30) NOT NULL,
  formatted_message TINYTEXT NOT NULL,
  message           TINYTEXT NOT NULL,
  logger            VARCHAR(255) NOT NULL,
  level             ENUM('trace', 'debug', 'info', 'warn', 'error', 'fatal') NOT NULL,
  plc_timestamp     TIMESTAMP NOT NULL,
  clock_timestamp   TIMESTAMP NULL,
  task_index        TINYINT NOT NULL,
  task_name         VARCHAR(63) NOT NULL,
  task_cycle_counter INT NOT NULL,
  app_name          VARCHAR(63) NOT NULL,
```

```

project_name      VARCHAR(63) NOT NULL,
onlinechange_count INT NOT NULL
);

CREATE TABLE IF NOT EXISTS log_argument (
  Id                BIGINT AUTO_INCREMENT PRIMARY KEY,
  log_entry_id      BIGINT NOT NULL REFERENCES log_entry(id),
  idx              TINYINT NOT NULL,
  value            TINYTEXT NOT NULL,
  type             ENUM('null', 'byte', 'word', 'dword', 'real', 'lreal', 'sint',
                        'int', 'dint', 'usint', 'uint', 'udint', 'string', 'bool', 'ularge', 'large') NOT NULL,
  UNIQUE (log_entry_id, idx)
);

CREATE TABLE IF NOT EXISTS log_context (
  id                BIGINT AUTO_INCREMENT PRIMARY KEY,
  log_entry_id      BIGINT NOT NULL REFERENCES log_entry(id),
  name             VARCHAR(255) NOT NULL,
  value            TINYTEXT NOT NULL,
  type             ENUM('null', 'byte', 'word', 'dword', 'real', 'lreal', 'sint',
                        'int', 'dint', 'usint', 'uint', 'udint', 'string', 'bool', 'ularge', 'large') NOT NULL,
  UNIQUE (log_entry_id, name)
);

```

## Postgres

### SimpleFlat

```

CREATE TYPE log_level_type AS ENUM('trace', 'debug', 'info', 'warn', 'error', 'fatal');

CREATE TABLE IF NOT EXISTS log_entry (
  id                BIGSERIAL PRIMARY KEY,
  source           VARCHAR(30) NOT NULL,
  hostname         VARCHAR(30) NOT NULL,
  formatted_message TEXT NOT NULL,
  logger          VARCHAR(255) NOT NULL,
  level           log_level_type NOT NULL,
  plc_timestamp    TIMESTAMP NOT NULL,
  clock_timestamp  TIMESTAMP NULL,
  task_index       SMALLINT NOT NULL,
  task_name        VARCHAR(63) NOT NULL,
  task_cycle_counter INT NOT NULL,
  app_name         VARCHAR(63) NOT NULL,
  project_name     VARCHAR(63) NOT NULL,
  onlinechange_count INT NOT NULL
);

```

```
);
```

## FullFlat

```
CREATE TYPE log_level_type AS ENUM('trace', 'debug', 'info', 'warn', 'error', 'fatal');
```

```
CREATE TABLE IF NOT EXISTS log_entry (  
  id                BIGSERIAL PRIMARY KEY,  
  source            VARCHAR(30) NOT NULL,  
  hostname          VARCHAR(30) NOT NULL,  
  formatted_message TEXT NOT NULL,  
  message           TEXT NOT NULL,  
  logger            VARCHAR(255) NOT NULL,  
  level             log_level_type NOT NULL,  
  plc_timestamp     TIMESTAMP NOT NULL,  
  clock_timestamp   TIMESTAMP NULL,  
  task_index        SMALLINT NOT NULL,  
  task_name         VARCHAR(63) NOT NULL,  
  task_cycle_counter INT NOT NULL,  
  app_name          VARCHAR(63) NOT NULL,  
  project_name      VARCHAR(63) NOT NULL,  
  onlinechange_count INT NOT NULL  
);
```

```
CREATE TYPE log_value_type AS ENUM('null', 'byte', 'word', 'dword', 'real', 'lreal', 'sint',  
'int', 'dint', 'usint', 'uint', 'udint', 'string', 'bool', 'ularge', 'large');
```

```
CREATE TABLE IF NOT EXISTS log_argument (  
  id                SERIAL PRIMARY KEY,  
  log_entry_id      BIGINT NOT NULL REFERENCES log_entry(id),  
  idx               SMALLINT NOT NULL,  
  value             TEXT NOT NULL,  
  type              log_value_type NOT NULL,  
  UNIQUE (log_entry_id, idx)  
);
```

```
CREATE TABLE IF NOT EXISTS log_context (  
  id                SERIAL PRIMARY KEY,  
  log_entry_id      BIGINT NOT NULL REFERENCES log_entry(id),  
  name              VARCHAR(255) NOT NULL,  
  value             TEXT NOT NULL,  
  type              log_value_type NOT NULL,  
  UNIQUE (log_entry_id, name)  
);
```

# SQL-Server

## SimpleFlat

```
CREATE TABLE log_entry (  
    id                INT IDENTITY PRIMARY KEY,  
    source            VARCHAR(30) NOT NULL,  
    hostname          VARCHAR(30) NOT NULL,  
    formatted_message TEXT NOT NULL,  
    logger            VARCHAR(255) NOT NULL,  
    level             CHAR(5) NOT NULL,  
    plc_timestamp     DATETIME2 NOT NULL,  
    clock_timestamp   DATETIME2 NULL,  
    task_index        SMALLINT NOT NULL,  
    task_name         VARCHAR(63) NOT NULL,  
    task_cycle_counter INT NOT NULL,  
    app_name          VARCHAR(63) NOT NULL,  
    project_name      VARCHAR(63) NOT NULL,  
    onlinechange_count INT NOT NULL  
);
```

## FullFlat

```
CREATE TABLE log_entry (  
    id                BIGINT IDENTITY PRIMARY KEY,  
    source            VARCHAR(30) NOT NULL,  
    hostname          VARCHAR(30) NOT NULL,  
    formatted_message TEXT NOT NULL,  
    message           TEXT NOT NULL,  
    logger            VARCHAR(255) NOT NULL,  
    level             CHAR(5) NOT NULL,  
    plc_timestamp     DATETIME2 NOT NULL,  
    clock_timestamp   DATETIME2 NULL,  
    task_index        SMALLINT NOT NULL,  
    task_name         VARCHAR(63) NOT NULL,  
    task_cycle_counter INT NOT NULL,  
    app_name          VARCHAR(63) NOT NULL,  
    project_name      VARCHAR(63) NOT NULL,  
    onlinechange_count INT NOT NULL  
);
```

```
CREATE TABLE log_argument (  
    id                BIGINT IDENTITY PRIMARY KEY,  
    log_entry_id      BIGINT NOT NULL,  
    idx              SMALLINT NOT NULL,
```

```

    value            TEXT NOT NULL,
    type             CHAR(6) NOT NULL,
    UNIQUE (log_entry_id, idx),
    FOREIGN KEY (log_entry_id) REFERENCES log_entry(id)
);

CREATE TABLE log_context (
    id                BIGINT IDENTITY PRIMARY KEY,
    log_entry_id      BIGINT NOT NULL,
    name              VARCHAR(255) NOT NULL,
    value            TEXT NOT NULL,
    type             CHAR(6) NOT NULL,
    UNIQUE (log_entry_id, name),
    FOREIGN KEY (log_entry_id) REFERENCES log_entry(id)
);

```

# Installationsanleitung von log4TC

## Setup

Den aktuellen Release von Log4TC kann [hier](#) geladen werden. Achten sie auf die Ziel Architektur x86 bzw x64!

## Voraussetzungen

- [TwinCat 3.1 \(min. 4022.00\)](#)
- Administrationsrechte für die Installation

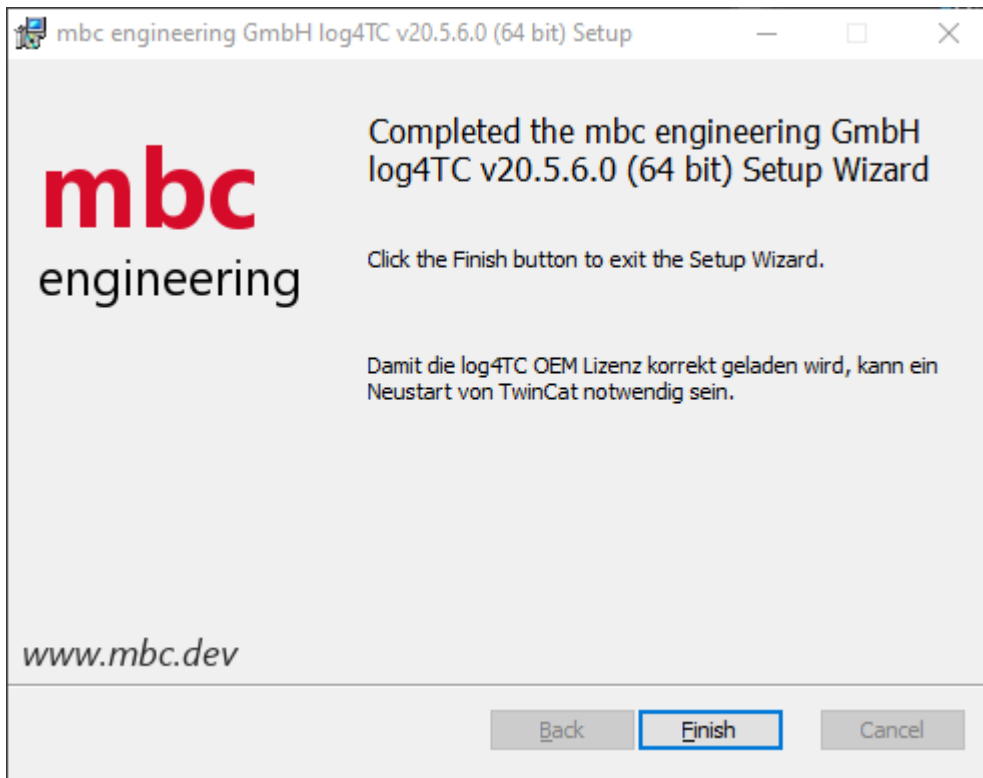
### Nur Service:

- min Windows 7 SP1 / Windows Embedded Standard 2009
- [Microsoft .NET Framework \(mindestens 4.6.1, empfohlen 4.8\)](#)
- [ADS Router - TC1000 | TC3 ADS](#)

## Beispiel Installation

Vorgehen zur Installation auf einem Zielsystem wie einem C6015 mit Windows 10 und einer x64 Architektur.

1. Stellen Sie sicher das alle Anwendungen geschlossen sind.
2. Kopieren des MSI [Mbc.Log4Tc.Setup\(x64\)v20.5.6.0](#) auf den Zielrechner. Führen Sie das MSI setup aus.
3. Akzeptieren Sie den **log4TC Software-Lizenzvertrag**.
4. Wählen sie die gewünschten Features. (Nähere Beschreibung [hier](#))
5. Durch Klicken auf **Install** werden alle notwendigen Dateien auf das System kopiert und der log4TC Windows Service mit dem Namen **mbc log4TC Service** gestartet.



## Features

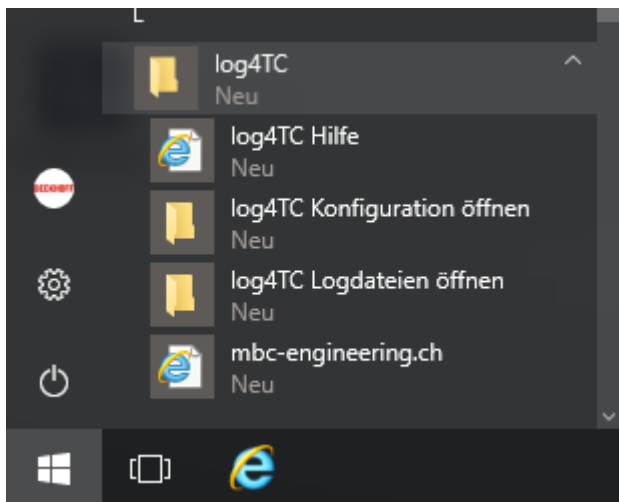
### log4TC Service

#### NOTE

Dieses Feature erscheint nur wenn sie ein ADS Router - TC1000 | TC3 ADS installiert haben.

#### Beinhaltet

- Windows Service zum schreiben der generierten Logmeldungen aus TwinCat
- Konfiguration Links im Startmenü





# log4TC TwinCat 3 Bibliothek

## NOTE

Dieses Feature erscheint nur wenn sie TwinCat 3.1 Engineering (XAE) min. 4022.00 installiert haben.

## Beinhaltet

- Installiert die log4TC Twincat 3 Bibliothek lokal
- Bereitet die OEM Lizenz zur Registrierung für die Produktive Benutzung vor
- Kopiert das getting started Projekt unter `C:\ProgramData\log4TC\gettingstarted`
- Hilfe Links im Startmenü

## Bekannte Fehler

**Setup endet mit dem Fehler: ... Setup Wizard ended prematurely because of an error. Your system has not been modified. ...**

In diesem Fall ist ein Fehler aufgetreten.



Starten sie das setup erneut mit der Kommandozeile ausgeführt als Administrator. Navigieren Sie in den Ordner mit dem MSI Setup per `cd [folder]`. Geben Sie folgendes ein: `msiexec.exe /i "[setup].msi"`

/l\*v install.log. Wenden Sie sich anschliessend mit dem `install.log` an uns.

# log4TC Servicekonfiguration

Der log4TC Service wird über eine JSON-Konfigurationsdatei im Pfad

%ProgramData%\log4TC\config\appsettings.json konfiguriert. Nach der Installation von log4TC wird eine Standardkonfiguration installiert, die alle Log-Meldungen auf NLog ausgibt.

Die Standardkonfiguration sieht wie folgt aus:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "Outputs": [
    {
      "Type": "nlog",
    }
  ]
}
```

Die Konfigurationsdatei wird auf Änderungen überwacht und automatisch neu geladen.

## Konfigurationsabschnitte

### Internes Logging

Über den Abschnitt **Logging** wird das log4TC interne Logging konfiguriert. Wir empfehlen hier die Standardkonfiguration beizubehalten. Interne Logs werden in die Datei

%ProgramData%\log4TC\internal\service.log geschrieben. Bei Problemen kann hier kontrolliert werden, ob der Service korrekt arbeitet.

### Ausgaben

Der Abschnitt **Outputs** enthält die Konfiguration aller Ausgaben. Es können ein oder mehrere Ausgaben konfiguriert werden. Das Schema ist für eine Ausgabe ist wie folgt:

```
"Outputs": [
  // Ausgabe 1
  {
    "Type": "Typ der Ausgabe",
    "Filter": {
```

```

        "Logger": "Logger-Muster",
        "Level": "Log-Level"
    },
    "ExcludeFilter": {
        "Logger": "Logger-Muster",
        "Level": "Log-Level"
    },
    "Config": {
        // Ausgabespezifisch
    }
},
// Ausgabe 2
{
    // ...
}
]

```

Mit **Type** wird die Ausgabe ausgewählt, wie z.B. **nlog**, **graylog**, **influxdb**. Ausgaben können mehrfach konfiguriert werden. Dieses Feld muss zwingend angegeben werden.

Mit **Filter** und **ExcludeFilter** können Log-Meldungen selektiert werden. Eine Log-Meldung muss den **Filter**-Kriterien entsprechen und nicht den **ExcludeFilter**-Kriterien, damit sie an die Ausgabe weitergeleitet wird. Das Kriterium **Logger** selektiert nach dem Loggernamen, es können wildcard (\*) Platzhalter am Anfang und Ende verwendet werden, wie z.B. **"Logger": "influxdb.\*"**. Mit **Level** wird der Loglevel geprüft, er muss gleich oder höher sein.

Mit dem Objekt **Config** können die Ausgaben spezifisch konfiguriert werden, die Inhalte können bei der Beschreibung der Ausgaben nachgelesen werden.

Siehe dazu die Dokumentation der Ausgaben:

- [NLog](#)
- [Graylog](#)
- [InfluxDb](#)