AS Logger

Strukturiertes Logging und Alarmaufzeichnung für CODESYS-SPS-Applikationen

Titel	AS Logger
Dokumenttyp	Anwenderhandbuch
Dateireferenz	AS_Logger.library
Version	2.2
Datum	17.08.2026
Autor	ASKS GmbH
Status	Freigegeben

ASKS GmbH
Tevesstraße 29a · 78176 Blumberg · www.as-ks.de · info@as-ks.de

Inhaltsverzeichnis

Zusammenfassung: Dieses Anwenderhandbuch beschreibt den ASKS Logger – eine CODESYS-Bibliothek für strukturiertes Logging und Alarmaufzeichnung. Es erklärt Einbindung, Initialisierung, das entkoppelte Schreiben über die Inbox sowie die Konfiguration und Registrierung von Sinks (inkl. FileSink-Rotation) und Decorators, die Filterung, Alarmprotokollierung, Fehlercodes und die Visualisierung.

1 Übersicht

Der ASKS Logger (AS_Logger) stellt strukturiertes Logging und Alarmaufzeichnung für CODESYS-SPS-Applikationen bereit. Ein Log-Ereignis wird über eine einheitliche API erzeugt, optional durch Decorators angereichert, durch Filter bewertet und an ein oder mehrere Ausgabeziele (Sinks) verteilt.
Die wichtigsten Funktionen der Bibliothek umfassen:

- Sechs Log-Level: TRACE, DEBUG, INFO, WARN, ERROR, FATAL
- Verkettbare Fluent-API mit strukturierten Schlüssel/Wert-Paaren und Platzhalter-Templates
- Alarmprotokollierung mit Schweregraden und Alarmcodes
- Mehrere gleichzeitige Sinks: Speicherpuffer für die Visualisierung, CODESYS-Log (als Standard-Komponente oder als eigene Log-Instanz), Dateien sowie Syslog nach RFC 5424 über UDP
- Automatische Ereignisanreicherung: Zeitstempel, Maschinen-/Host-Kontext, Quellkontext, Maskierung sensibler Felder
- Filterung pro Sink
- Offene Schnittstellen (ISink, IDecorator, ISinkFilter) für eigene Erweiterungen ohne Eingriff in den Kern

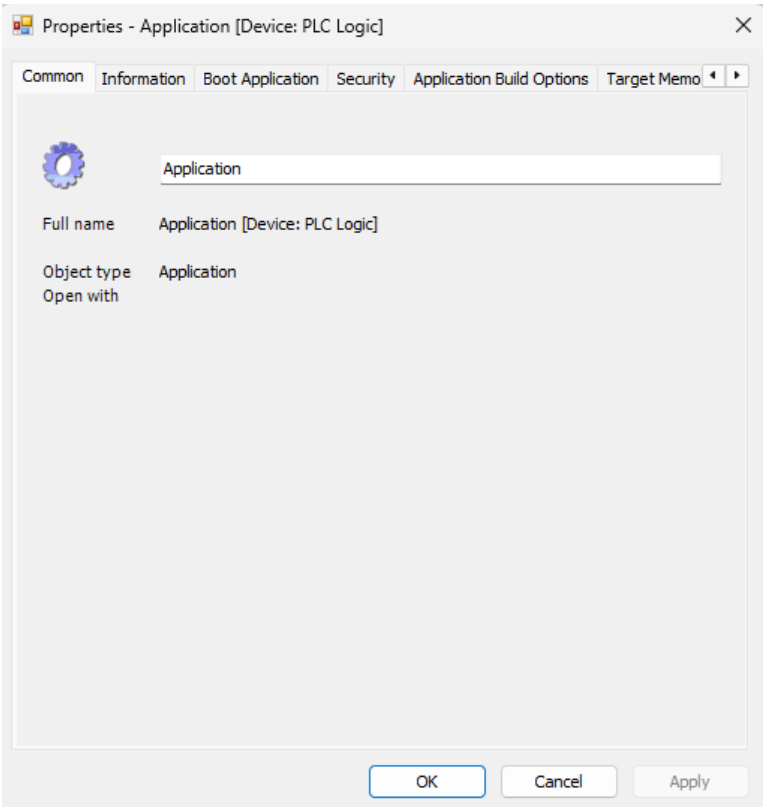
Das Schreiben von Log-Events ist von der Ausgabe entkoppelt: Ein Log-Aufruf legt das Ereignis lediglich in eine Inbox ab und kehrt sofort zurück. Eine eigene Worker-Task, die der Logger selbst anlegt, arbeitet die Inbox ab – ein zyklischer Aufruf durch die Applikation ist dafür nicht erforderlich. Die Inbox fasst standardmäßig 100 Einträge; bei Überlauf werden die ältesten Einträge verworfen (Ringpuffer-Verhalten).
Die Bibliothek benötigt die Bibliothek AS_Util (UTL) (z. B. Semaphore, System-Stringparameter).

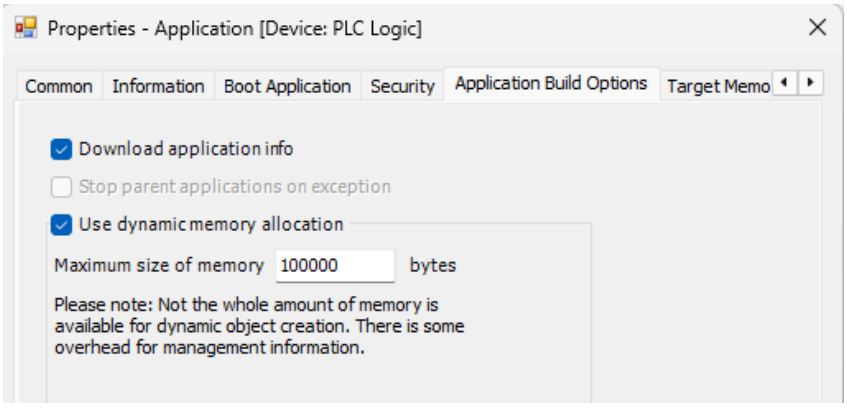
2 Anleitung

2.1 Applikation

Damit die Logger-Funktionen genutzt werden können, muss die Bibliothek AS_Logger im Library Manager eingebunden werden. Die Bibliothek AS_Util (UTL) wird als Abhängigkeit benötigt.

Der Logger nutzt dynamischen Speicher, daher muss der dynamische Speicher der Applikation vergrößert werden: unter Application → Properties → Application Build Options die Option "Use dynamic memory allocation" aktivieren und die maximale Größe erhöhen.





Hinweis: Ohne die Erweiterung des dynamischen Speichers kann der Logger nicht initialisiert werden.

2.2 Logger

Eine lokale Logger-Instanz wird über den Typ LOC.Logger deklariert. Innerhalb einer lokalen Instanz werden Sinks, Decorators und Filter ohne Namespace-Präfix angesprochen. Wird die Instanz als globale Referenz registriert, erfolgt der Zugriff von überall über LOC.UTL.Logger (statische Aufrufe) bzw. LOC.UTL.logger (Fluent-Aufrufe).
Die Mindest-Level-Prüfung erfolgt vor dem Aufbau des Ereignisses, sodass Ereignisse unterhalb des konfigurierten Levels nur minimale Last erzeugen.

```
// --- Minimum level ---

// Set global minimum log level
eErr := LOC.UTL.logger.SetMinLevel(eLevel := LOC.UTL.LOG_LEVEL.DEBUG);
```

Das Schreiben ist von der Ausgabe entkoppelt: Ein Log-Aufruf legt das Ereignis lediglich in die Inbox des Loggers ab und kehrt sofort zurück. Eine eigene Worker-Task, die der Logger selbst anlegt, arbeitet die Inbox ab – langsame Sinks (z. B. Syslog über UDP) blockieren die aufrufende Task dadurch nie.

2.2.1 Globale Logger-Referenz

Die lokale Logger-Instanz kann einmalig als globale Referenz registriert werden, sodass der Logger aus jeder POU ohne Weitergabe der Instanz erreichbar ist:

```
// === Logger ===

lo   : LOC.Logger;
```

```
// --- Set global logger reference ---

// Set local Logger as global Logger reference
LOC.UTL.Logger.SetLogger(lo);
```

Nach der Registrierung erfolgt jeder weitere Zugriff — die Konfiguration von Sinks, Decorators und Filtern ebenso wie sämtliche Log- und Alarmaufrufe — über LOC.UTL.Logger (statische Aufrufe, z. B. SetLogger, Diagnosezähler) bzw. LOC.UTL.logger (Fluent-Aufrufe wie SetKey, Source, Info, LogAlarm). Die lokale Instanz wird danach nicht mehr direkt verwendet.

2.2.2 Strukturiertes Logging und Platzhalter

Mit SetKey übergebene Schlüssel/Wert-Paare werden in passende {key}-Platzhalter des Nachrichten-Templates eingesetzt. Source setzt die Ereignisquelle. Die Level-Methoden (Trace,

Debug, Info, Warn, Err, Fatal) liefern eine Referenz auf den Logger zurück und sind daher verkettbar.

```
LOC.UTL.logger
    .SetKey(sKey := 'speed', sVal := '450')
    .SetKey(sKey := 'time',  sVal := '00:12:30')
    .Source(sSource := __POUNAME())
    .Info(sTemplate := 'Granulation running, mixer speed {speed} rpm, elapsed
{time}');
```

Soll eine Sequenz nicht durch andere Tasks unterbrochen werden, wird die Kette in Lock/Unlock eingeschlossen:

```
LOC.UTL.logger
    .Lock()
    .SetKey(sKey := 'session', sVal := 'sess-batch-88')
    .SetKey(sKey := 'action',  sVal := 'emergency_stop')
    .Source(sSource := __POUNAME())
    .Err(sTemplate := 'Critical fault in session {session} during {action}')
    .Unlock();
```

2.3 Sinks

Ein Sink ist ein Ausgabeziel, das die Schnittstelle ISink implementiert. Sinks erben von SinkBase, das vor der eigentlichen Ausgabe (DoEmit) das sink-eigene Mindest-Level, den zugewiesenen Filter und lokale Decorators anwendet. Jeder Sink wird unabhängig über AddDecorator, SetFilter und SetMinLevel konfiguriert.

Sink	Zweck
MemorySink	Ringpuffer im Speicher für Anzeige/Visualisierung (GetCount, GetAt, UpdateVisu)
CodesysLogSink	Schreibt Einträge als eigene Komponente in das Standard-CODESYS-Log (SetComponent legt den Komponentennamen fest; das Log-Level wird auf die passende CODESYS-Log-Klasse gemappt)
CmpLogSink	Erstellt eine eigene, neue CODESYS-Log-Instanz (statt in eine Komponente des Standard-CODESYS-Logs zu schreiben) und schreibt die Einträge dort hinein. Die Instanz kann im Log-Viewer über das Dropdown-Feld rechts oben ausgewählt werden (Init).

SyslogUdpSink	Syslog RFC 5424 über UDP (Init mit Server-IP/Port, Hostname, Anwendungsname, Facility, PEN)
FileSink	Schreibt Ereignisse als logfmt-Zeilen in eine Datei auf der Steuerung über AS_File — keine Runtime-Log-Komponente (CmpCodesysLog) erforderlich. Größenbasierte Rotation mit konfigurierbarer Dateigröße und Archivanzahl (Init).

Tabelle 1: Verfügbare Sinks

```
// === Sinks ===

plcLogSink    : CodesysLogSink;
memSink       : MemorySink;
filesink      : FileSink;
syslogSink    : SyslogUdpSink;
CmpLogSink    : CmpLogSink;
```

```
// --- Sink configuration ---

// Create new component in CODESYS log
eErr := plcLogSink.SetComponent(sComponent := 'LoggerRecipeDemo');

eErr := syslogSink.Init(
            // Init UDP sink
            sServerIp   := '192.168.100.77',
            // Local network adapter IP used to send
            sLocalIp    := '192.168.100.111',
            // Hostname reported in the syslog message
            sHostname   := 'PLC01',
            // Application name reported in the syslog message
            sAppName    := 'LoggerRecipeDemo');

eErr := filesink.Init(
            // Name of the file (base name, without extension)
            sName       := 'Test',
            // Directory to store files in
            sDir        := 'PlcLog',
            // File type / extension
            sFileType   := 'txt',
            // Max size per file in bytes (1 MB)
            nMaxBytes   := 1048576,
            // Max number of rotated files
            nMaxFiles   := 5);
```

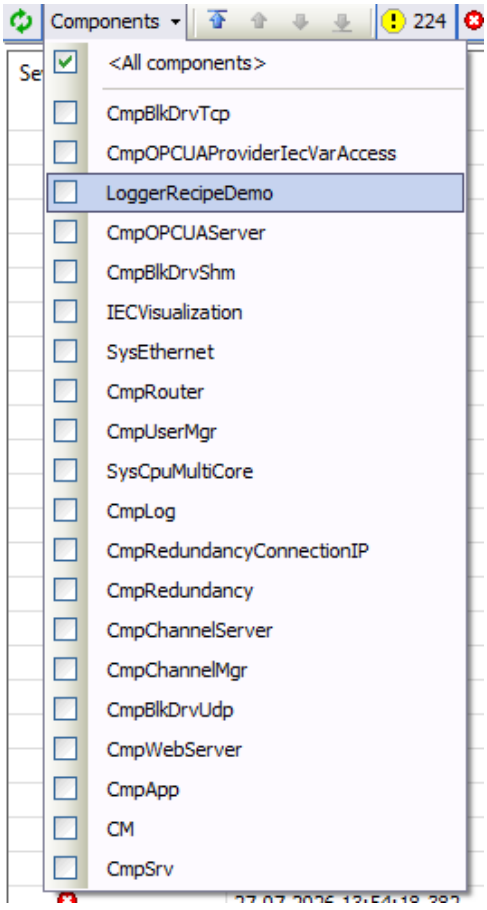
```
eErr := CmpLogSink.Init(
            // Create new Logger instance in CODESYS Device Log
            sName       := 'TestCMP',
            // Name of the component
            sComponent  := 'LoggerDemo');

// Number of visible MemorySink lines
eErr := memSink.Init(nVisuLines := 30);
```

2.3.1 CodesysLogSink

SetComponent() legt eine eigene Komponente im Standard-CODESYS-Log an, unter der die Einträge erscheinen.

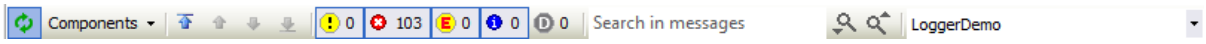
```
// Create new component in CODESYS log
eErr := plcLogSink.SetComponent(sComponent := 'LoggerRecipeDemo');
```



2.3.2 CmpLogSink

Anders als die CodesysLogSink erstellt die CmpLogSink eine eigene, neue Log-Instanz, die im Log-Viewer über das Dropdown-Feld rechts oben ausgewählt werden kann.

```
eErr := CmpLogSink.Init(
            // Create new Logger instance in CODESYS Device Log
            sName       := 'LoggerDemo',
            // Name of the component
            sComponent  := 'AppLog');
```



2.3.3 FileSink

Die FileSink schreibt jedes Ereignis als logfmt-Zeile in eine Datei im Dateisystem der Steuerung (über FIL.FileAscii) und benötigt dafür — anders als CodesysLogSink/CmpLogSink — keine Runtime-Log-Komponente (CmpCodesysLog):

```
2026-07-23-12:34:56.789 INFO Belt started user=jj pump=3
```

Init() legt bei Bedarf das Zielverzeichnis an und übernimmt Dateiname, Typ sowie Rotationsgrenzen. Ist sDir oder sName leer, liefert Init ERR_WRONG_INPUT.
Größenbasierte Rotation: Vor jedem Schreiben prüft die FileSink, ob die aktuelle Dateigröße (aus FIL.FileAscii.nSize gelesen, auch nach einem Neustart korrekt) zusammen mit der neuen Zeile nMaxBytes überschreiten würde. Ist das der Fall, wird zuerst rotiert: Die älteste Archivdatei (Index nMaxFiles) wird gelöscht, alle verbleibenden Archive werden hochgezählt (.i → .(i+1)), und die aktuelle Datei wird zum Archiv mit Index 1.
Archivdateien werden dabei nach dem Muster <Name>_<Index>.<Typ> benannt, z. B. Test_1.txt, Test_2.txt … bis Test_5.txt bei nMaxFiles := 5. Die Datei bleibt zwischen den Schreibzugriffen offen und wird nur bei Rotation bzw. im FB_Exit geschlossen. Datei-I/O-Fehler werden als ERR_LOG_RUNTIME_FAILURE gemeldet.

2.3.4 Sinks registrieren

Ein konfigurierter Sink wird erst aktiv, wenn er zusätzlich beim Logger registriert wird — ohne diesen Schritt bleiben Init, SetFilter & Co. wirkungslos, da der Sink nie ein Ereignis von Emit erhält:

```
// --- Register all sinks ---

// Register CODESYS log sink
eErr := LOC.UTL.logger.AddSink(sink := plcLogSink);
// Register memory sink
eErr := LOC.UTL.logger.AddSink(sink := memSink);
// Register file sink
eErr := LOC.UTL.logger.AddSink(sink := filesink);
// Register cmp sink
eErr := LOC.UTL.logger.AddSink(sink := CmpLogSink);
```

AddSink liefert ERR_LOG_SINK_LIMIT_REACHED, wenn die maximale Anzahl Sinks (GV_LOG.c_nMaxSinks) bereits erreicht ist.

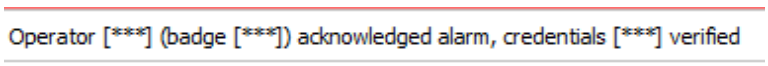
2.4 Decorators

Ein Decorator reichert ein Ereignis vor der Ausgabe an (Schnittstelle IDecorator). Decorators können global am Logger (logger.AddDecorator) oder pro Sink (sink.AddDecorator) hinzugefügt werden. Ein sink-lokaler Decorator arbeitet auf einer Kopie des Ereignisses und beeinflusst andere Sinks nicht.

Decorator	Zweck
MachineDecorator	Ergänzt Maschinen-/Host-Kontext — Hostname, Seriennummer, Firmware-Version, Standort — über SetConfig
SourceContextDecorator	Ergänzt einen frei definierbaren Quellkontext über SetSource (z. B. zur Herkunftskennzeichnung einzelner Sinks)
RedactionDecorator	Maskiert sensible Schlüssel im Payload über AddKey mit einem konfigurierbaren Platzhalter (SetReplacement)

Tabelle 2: Verfügbare Decorators

Beispiel: Sensible Schlüssel nur auf dem Syslog-Sink maskieren — Konsole und Speicher zeigen die echten Werte, Syslog zeigt "[***]".



Wie bei den Sinks gilt auch für Decorators: Erst die Registrierung macht sie wirksam — global am Logger für alle Sinks, oder lokal an einem einzelnen Sink:

```
// --- Global decorators ---

// Set decorator for all sinks
eErr := LOC.UTL.logger.AddDecorator(decorator := machineD);

// --- Sink-local decorator ---

// Set decorator for FileSink
eErr := filesink.AddDecorator(decorator := redactor);
```

Globale Decorators laufen vor sink-lokalen. AddDecorator liefert ERR_LOG_DECORATOR_LIMIT_REACHED (global begrenzt durch GV_LOG.c_nMaxGlobalDec), wenn das jeweilige Limit erreicht ist.

RedactionDecorator.SetReplacement hat als Standardwert [REDACTED], solange kein eigener Platzhalter gesetzt wird (im Beispiel oben auf [***] geändert).
SourceContextDecorator.SetSource wird mit einem leeren String zu einem No-op (keine Quellkennzeichnung).

2.5 Filter

Ein Filter entscheidet, ob ein Ereignis angenommen wird (Schnittstelle ISinkFilter, Accept liefert BOOL). Filter werden pro Sink über SetFilter konfiguriert.

Filter	Zweck
LevelFilter	Lässt Ereignisse innerhalb eines Mindest-/Höchst-Levels zu (SetMinLevel, SetMaxLevel)
AlarmSeverityFilter	Nimmt an/verwirft anhand einer Schweregrad-Maske; kann eine Alarm-Eigenschaft erfordern (SetSeverityMask, AcceptSeverity, RejectSeverity, SetRequireAlarmProperty, Clear)

Tabelle 3: Verfügbare Filter

LOC.UTL.ALARM_SEVERITY ist als Bitmaske aufgebaut (NONE=1, INFO=2, WARN=4, STOP=8, ABORT=16, REBOOT=32), sodass sich beliebige Kombinationen von Schweregraden als ODER-Verknüpfung abbilden lassen. Über AcceptSeverity/RejectSeverity werden einzelne Schweregrade freigegeben bzw. ausgeschlossen, über SetSeverityMask wird die gesamte Maske in einem Aufruf gesetzt. Mit SetRequireAlarmProperty(xRequire := TRUE) werden alle Einträge ohne Alarmschwere verworfen — nützlich für Sinks, die ausschließlich Alarme führen sollen.
LevelFilter:

```
// --- Sink Level Filter: only info and above ---

// Set min level to INFO and above
eErr := levelFilter.SetMinLevel(eLevel := LOC.UTL.LOG_LEVEL.INFO);
// Set filter for MemorySink
eErr := memSink.SetFilter(filter := levelFilter);
// Set filter for FileSink
eErr := filesink.SetFilter(filter := levelFilter);
```

AlarmFilter:

```
// --- Sink Alarm Filter: ---

eErr := alarmFilter.AcceptSeverity(eSeverity := LOC.UTL.ALARM_SEVERITY.STOP);
eErr := alarmFilter.AcceptSeverity(eSeverity := LOC.UTL.ALARM_SEVERITY.ABORT);
eErr := alarmFilter.AcceptSeverity(eSeverity := LOC.UTL.ALARM_SEVERITY.REBOOT);
// Set filter for CmpLogSink
```

```
eErr := CmpLogSink.SetFilter(filter := alarmFilter);
```

2.6 Alarme

Alarme werden mit LogAlarm protokolliert; übergeben werden ein LOC.UTL.ALARM_SEVERITY, ein Alarmcode und ein Nachrichten-Template. Das Hilfsprogramm AlarmMapping stellt Konvertierungen zwischen Alarm-Schweregrad und Log-Level bereit (ToLogLevel, StringToSeverity).

```
LOC.UTL.logger
    .SetKey(sKey := 'temp', sVal := '132.8')
    .Source(sSource := __POUNAME())
    .LogAlarm(
        eSeverity  := LOC.UTL.ALARM_SEVERITY.STOP,
        nAlarmCode := 3001,
        sTemplate  := 'Sterilization temp {temp}°C exceeds critical max, aborting
cycle'
    );
```

2.7 Diagnose

Der Logger stellt Diagnosezähler bereit: nTotalEvents (alle erzeugten Ereignisse), nDroppedByMinLevel (durch die Mindest-Level-Prüfung verworfen) und nWritten (tatsächlich geschriebene Ereignisse). ResetCounters setzt die Zähler zurück.

```
// --- Diagnostics ---

// Current event counter
nEvents         := LOC.UTL.Logger.nTotalEvents;
// Dropped events
nDroppedEvents  := LOC.UTL.Logger.nDroppedByMinLevel;
// Actual written events
nWritten        := LOC.UTL.Logger.nWritten;
// Reset all counters
LOC.UTL.Logger.ResetCounters();
```

2.8 Fehlercodes

Alle Methoden liefern einen Wert vom Typ LOC.ERROR zurück. Neben allgemeinen Basis-Codes (u. a. ERR_ALREADY_INIT, ERR_WRONG_INPUT, ERR_NOT_FOUND, ERR_OVERFLOW) definiert AS_Logger folgende spezifischen Codes:

Code	Bedeutung
ERR_LOG_SINK_LIMIT_REACHED	Maximale Anzahl Sinks (GV_LOG.c_nMaxSinks) bereits über AddSink registriert

ERR_LOG_DECORATOR_LIMIT_REACHED	Maximale Anzahl Decorators (global oder sink-lokal) bereits über AddDecorator registriert
ERR_LOG_PROP_LIMIT_REACHED	Maximale Anzahl Schlüssel/Wert-Paare (SetKey) für ein Ereignis erreicht
ERR_LOG_REDACTION_KEY_LIMIT_REACHED	Maximale Anzahl maskierter Schlüssel im RedactionDecorator (AddKey) erreicht
ERR_LOG_LEVEL_BELOW_MIN	Ereignis liegt unterhalb des konfigurierten Mindest-Levels, von Emit verworfen
ERR_LOG_FILTER_REJECTED	Ereignis vom zugewiesenen Filter (SetFilter) abgelehnt
ERR_LOG_NULL_REFERENCE	Sink nicht initialisiert (z. B. FileSink ohne vorherigen Init-Aufruf)
ERR_LOG_RUNTIME_FAILURE	Laufzeitfehler bei der Ausgabe, z. B. Datei-I/O-Fehler in der FileSink
ERR_LOG_CFG_LOGGER_MISSING	Zugriff über LOC.UTL.Logger, bevor SetLogger eine lokale Instanz als globale Referenz registriert hat

Tabelle 4: Fehlercodes (LOC.ERROR)

3 Visualisierung

3.1 Vorbereitungen

Der MemorySink hält die letzten Ereignisse in einem Ringpuffer, der an eine Visualisierung gebunden werden kann. UpdateVisu wird zyklisch aufgerufen (z. B. im Hauptprogramm oder in der VISU_TASK), um die angezeigten Einträge zu aktualisieren.
Zum Anzeigen wird die Visualisierung VisuLoggerList in einem Frame instanziiert und der MemorySink übergeben. Der Scaling Type des Frames ist für eine korrekte Anzeige auf Fixed zu stellen. Die Anzahl der angezeigten Zeilen kann je nach Bildschirmgröße angepasst werden (1–50).

```
// Number of visible MemorySink lines
eErr := memSink.Init(nVisuLines := 30);
```

Property	Value
Element name	LoggerList
Type of element	Frame -> LOC.VisuLoggerList
Scroll Bar size	From style
Clipping	☐
Show frame	No frame
Scaling type	Fixed
Swiping behavior	Not swipable
Swiping preview	☑
References	Configure...
LOC.VisuLoggerList	0
source	Main.memSink

3.2 Speicherpuffer

Die im MemorySink gepufferten Ereignisse werden über die VisuLoggerList als scrollbare Liste mit Zeitstempel, Level und Nachricht dargestellt.

timestamp	severity	message
27.07.2026 13:54:30.004	error	Critical fault in session sess-batch-88 during emergency_stop
27.07.2026 13:54:29.496	error	Sterilization temp 132.8°C exceeds critical max, aborting cycle
27.07.2026 13:54:28.992	warning	Sterilization temp 121.4°C nearing limit 125.0°C
27.07.2026 13:54:28.486	info	Recipe write acknowledged, non critical
27.07.2026 13:54:27.980	info	Operator jjones (badge 4711) acknowledged alarm, credentials Sunshine2024! verified
27.07.2026 13:54:27.474	error	Mixer speed 1150 rpm exceeds critical threshold, controlled stop
27.07.2026 13:54:26.968	warning	Mixer speed 980 rpm approaching limit 1000 rpm

4 Sicherheitshinweise

Hinweis: Entwurf. Dieser Abschnitt entsteht im Rahmen der CRA-Vorbereitung (Maßnahmen M-19 und M-21) und ist noch nicht freigegeben. Vor der Veröffentlichung sind die Aussagen zum Auslieferungszustand gegen den tatsächlichen Code und die Beispielprojekte zu prüfen.

Die folgenden Hinweise richten sich an Integratoren und Betreiber. Sie benennen die bestimmungsgemäße Verwendung sicherheitsrelevanter Funktionen sowie die verbleibenden Restrisiken, die in der Verantwortung des Integrators liegen.

4.1 Auslieferungszustand

Der Logger überträgt und speichert von sich aus keine Daten. Jede Senke muss in der Applikation explizit deklariert, initialisiert und dem Logger hinzugefügt werden. Es ist damit eine bewusste Entscheidung des Integrators, ob Log-Daten die Steuerung verlassen und auf welchem Weg.

4.2 Netzwerkübertragung über SyslogUdpSink

Die SyslogUdpSink überträgt Log-Einträge nach RFC 5424 über UDP. Das Protokoll arbeitet konstruktionsbedingt unverschlüsselt und ohne Authentifizierung; es gibt weder Vertraulichkeits- noch Integritätsschutz, und der Empfänger kann die Herkunft einer Nachricht nicht überprüfen. Log-Daten können Rückschlüsse auf Prozessparameter, Betriebszustände und Anlagenstruktur zulassen.
Bestimmungsgemäße Verwendung: Übertragung innerhalb eines vertrauenswürdigen, segmentierten Netzes, in dem Sender und Empfänger unter derselben administrativen Kontrolle stehen.

Hinweis: Die SyslogUdpSink ist nicht für den Einsatz über Netzsegmentgrenzen hinweg, über Funkstrecken oder über öffentliche Netze vorgesehen. Bestehen erhöhte Anforderungen an Vertraulichkeit oder Integrität, ist ein lokaler Syslog-Relay im geschützten Segment vorzusehen, der die Weiterleitung verschlüsselt (z. B. Syslog über TLS). AS_Logger stellt selbst keine verschlüsselte Senke bereit.

4.3 Inhalte von Log-Einträgen

Der Logger bewertet die übergebenen Inhalte nicht. Welche Daten protokolliert werden, entscheidet die Applikation. Nicht protokolliert werden sollten insbesondere Zugangsdaten und Schlüsselmaterial, personenbezogene Daten über das für den Betrieb erforderliche Maß hinaus sowie schützenswertes Prozess-Know-how wie vollständige Rezepturen.
Für Fälle, in denen sensible Schlüssel im Payload unvermeidbar sind, steht der RedactionDecorator zur Verfügung. Er kann sink-lokal eingesetzt werden, sodass Werte beispielsweise in der Visualisierung im Klartext erscheinen, auf einer Netzwerksenke aber maskiert werden. Die Maskierung greift nur für die über AddKey registrierten Schlüssel — die Liste ist bei Änderungen an der Applikation nachzuführen.

4.4 Dateiausgabe über FileSink

Die FileSink schreibt Log-Dateien in das Dateisystem der Steuerung. Der Schutz dieser Dateien gegen unbefugtes Lesen richtet sich nach den Zugriffsrechten der Steuerung und liegt in der Verantwortung des Betreibers. Bei der Wahl von Dateigröße und Archivanzahl ist zu berücksichtigen, wie lange Log-Daten auf dem Gerät verbleiben und wer auf das Gerät zugreifen kann.

4.5 Meldung von Sicherheitsproblemen

Hinweise auf mögliche Sicherheitslücken in dieser Bibliothek nehmen wir entgegen und behandeln sie nach unserer Richtlinie zur koordinierten Offenlegung. (Kontaktadresse und Link zur Richtlinie werden ergänzt, sobald die zugehörigen Maßnahmen abgeschlossen sind.)

5 Lizenz

Informationen zur Lizenzierung dieser Bibliothek erhalten Sie bei ASKS GmbH. Bitte kontaktieren Sie uns unter support@as-ks.de oder über unsere Website www.as-ks.de.