

Projektarbeit im Studiengang Informationstechnik

Projektarbeit

T2_2000

Bearbeitungszeit: 10.01.2022 - 15.09.2022

vorgelegt von: **Bernd Dorer**

Matrikelnummer: 4434433

Kurs: TINF20B3

E-Mail: bernd.dorer@rena.com

Abgabedatum: 15.09.2022



Betreuer:

Dipl.-Ing. (FH)

Heiko Schirmaier

RENA Technologies GmbH

Höhenweg 1

78148 Gütenbach

Tel.: +49 7723 9313-680

E-Mail: heiko.schirmaier@rena.com



Duale Hochschule
Baden-Württemberg
Karlsruhe

DHBW Karlsruhe

Erzbergerstraße 121

76133 Karlsruhe

Tel.: +49 721 9735-5

E-Mail: info@dhw-karlsruhe.de

Erklärung

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Aus den benutzten Quellen direkt oder indirekt übernommene Gedanken habe ich als solche kenntlich gemacht.

Diese Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, Datum

Unterschrift

Sperrvermerk

Sperrvermerk

Die nachfolgende Arbeit enthält vertrauliche Daten und Informationen der RENA Technologies GmbH. Veröffentlichungen oder Vervielfältigungen - auch nur auszugsweise - sind ohne ausdrückliche schriftliche Genehmigung des Unternehmens nicht gestattet. Die Arbeit ist nur den Korrektoren sowie den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

Versionshistorie

Version	Autor	Datum	Beschreibung
1.0	Bernd Dorer	10.01.22	Erstellung des Dokumentes
1.1	Bernd Dorer	21.03.22	Erstellung Aufgabenstellung
1.2	Bernd Dorer	22.03.22	Erstellung Recherche, TCP-Logging, CSV-Service
1.3	Bernd Dorer	25.03.22	Überarbeitung der erstellten Teile
2.0	Bernd Dorer	29.08.22	Erstellung des SPS-Logging
2.0	Bernd Dorer	06.09.22	Freigabe zum Korrekturlesen
2.1	Bernd Dorer	13.09.22	Korrekturen

Abbildungsverzeichnis

1	Automatisierungspyramide	3
2	Lasten-/Pflichtenheft für „TCP Logging“	4
3	Lasten-/Pflichtenheft für „CSV Service“	5
4	Lasten-/Pflichtenheft für „SPS-Logging“	6
5	RENA-Maschine „RENA EPM 2 100F“	7
6	Projektplan TCP-Logging	12
7	Projektplan CSV-Reports	13
8	Projektplan SPS-Logging	14
9	Navigations-WPF einer RENA-Maschine	16
10	Funktionen und Funktions-Baustein aus dem TIA-Portal	17
11	Telegramm des Consumption-Loggings	18
12	Telegramm des Process-Value-Loggings	18
13	Verbindungsauflbau einer TCP-Kommunikation	18
14	Alle unterstützten Telegramme und Datenbank-Tabellen	19
15	API-Guide für AGLink Bibliothek	21
16	Programmablaufplan TCP-Logging	23
17	OSI-Modell für Netzwerkprotokolle	24
18	Telegramme-Aufbau	31
19	Tabelle für das Consumption-Logging	34
20	Programmablaufplan SPS-Logging	41
21	ACCON-AGLink Kommunikationskonfigurator	42
22	Datenbaustein mit Beispielvariablen	50
23	Maschinenkonfiguration für ein QDR-Modul	51
24	Beispiel Tabelle	61

Abkürzungsverzeichnis

SPS Speicherprogrammierbare Steuerung

WPF *Windows Presentation Foundation*

DLL *Dynamic Link Library*

UTC *Coordinated Universal Time* (koordinierte Weltzeit)

Codelisting-Verzeichnis

1	Initialisierung des Sockets (C#)	22
2	Callback auf Socket starten (C#)	24
3	Callback, wenn TCP-Daten empfangen wurden (C#)	25
4	RunReceive-Thread (C#)	26
5	Daten-Konvertierung Bytes zu String (C#)	27
6	Switch-Case zur Auswahl der Korrekten Konvertierungsfunktion (C#)	29
7	Konvertierung eines Dosage-Telegramm (C#)	30
8	Abbildungs-Objekt der Consumption-Tabelle (C#)	32
9	Create-Script für eine Consumption-Tabelle (SQL)	32
10	Create-Script für eine Sequence (SQL)	33
11	Write-Funktion des Consumption-Logging (C#)	33
12	Create-Script für die Trigger-Funktion (SQL)	35
13	Script für die Zuweisung der Trigger-Funktion zur Tabelle (SQL)	36
14	Funktion CreateCSVReport, welche die Reports erstellt (C#)	37
15	Funktion zum schreiben aller Prozess-Schritt-Daten (C#)	38
16	Funktion zum schreiben der Dosage-Daten (C#)	39
17	Initialisierung der PLC-Communikation (C#)	40
18	Öffnen eines Projekts von der Speicherprogrammierbare Steuerung (SPS) (C#)	43
19	Zuweisen der Verbindungsparameter (C#)	44
20	Verbindung aufbauen (C#)	45
21	Struct für das Lesen einer Variable (C#)	46
22	Buffer erstellen (C#)	47
23	Lesen der Symbole (C#)	49
24	Erstellen des CreateTable-Befehl (C#)	52
25	Erstellen der CreateColumn-Befehls (C#)	53
26	Überprüfung der SQL-Tabelle (C#)	54
27	Überprüfung der SQL-Tabelle (SQL)	54
28	Überprüfung der SQL-Spalten (SQL)	55
29	Daten in Datenbank schreiben (C#)	56
30	Empfangene Daten von der Visualisierung auswerten (C#)	58
31	Daten per TCP an Visualisierung senden (C#)	59

Inhaltsverzeichnis

1 Einleitung	1
1.1 RENA Technologies GmbH	1
2 Aufgabenstellung	2
3 Konzept	9
3.1 Logging	9
3.2 Anwendung	9
3.3 Datenspeicherung	10
3.4 Projektplan	11
4 Recherche	15
4.1 .NET	15
4.2 WPF	15
4.3 C#	15
4.4 Telegramme	17
4.5 TCP	20
4.6 PostgreSQL	20
4.7 Kommunikationsbibliothek (.NET ↔ SIEMENS-SPS)	20
5 TCP-Logging	22
5.1 TCP-Kommunikation	22
5.2 Datenkonvertierung	26
5.3 SQL-Kommunikation	31
6 RENA CSV Service	35
6.1 SQL-Kommunikation	35
6.2 Auswertung	36
6.3 CSV-Datei	38
7 SPS-Logging	40
7.1 SPS-Zugriff	40
7.1.1 AGLink Kommunikationsbibliothek	40
7.1.2 SIEMENS-Variablen	49
7.2 SQL-Kommunikation	52
7.3 HMI-Kommunikation	57
7.3.1 Visualisierung zu Service	57
7.3.2 Service zu Visualisierung	59

8 Ausblick	60
8.1 TCP-Logging	60
8.2 SPS-Logging	60
9 Literatur	62
10 Anhang	63
10.1 RENA CSV Report als CSV-Datei	63
10.2 RENA CSV Report als Excel-Datei	64

1 Einleitung

1.1 RENA Technologies GmbH

Die Firma RENA Technologies GmbH, mit dem Hauptsitz in Gütenbach im Schwarzwald, ist ein auf nasschemische Maschinen spezialisierter Maschinenbauer, der 1993 gegründet wurde. Seither wurden von dem Maschinenhersteller mehr als 3100 Maschinen in 7 Produktionsstandorten und 5 Innovationszentren entwickelt und installiert. Mit automatisierten Maschinen für die Branchen Halbleiter, Medizintechnik, erneuerbare Energien, Glas und *additive Manufacturing* ist RENA der weltweite Technologieführer im Bereich nasschemische Maschinen. Dank seiner mehr als 130 Ingenieuren und 20 Servicestandorte weltweit setzt RENA darauf, ihre Maschinen zu einer langfristigen Erfolgsgeschichte zu machen. Mit Laboren in Deutschland, Österreich, China und den USA konnte RENA schon über 200 Patente und Patentanmeldungen für neue Innovationen erzielen.

„Mit Leidenschaft, Kundennutzen durch Innovation, Technologieführerschaft und operative Stärke zu erschaffen und somit einen gefestigten Stand als Weltmarktführer für nasschemische Produktionsmaschinen in unterschiedlichen Branchen zu erlangen“, das ist das Leitbild der RENA.

Eingesetzt wurde ich im Hauptsitz in Gütenbach in der Abteilung Software Development.

2 Aufgabenstellung

Die RENA Maschinen generieren während der Produktion eine Vielzahl von Daten, welche für den Kunden, aber auch für den RENA Service wichtig sind. Hierzu wurde das Projekt „RENA Data Logging“ ins Leben gerufen, welches die bisherige Logging-Software ersetzen und verbessern soll.

Die Automatisierungspyramide (Abbildung 1) stellt die verschiedenen Ebenen der Automatisierung dar. Auf Ebene 0 ist die Fertigung an sich dargestellt. In der Feldebene ist die Sensorik und Aktorik eingeordnet und auf der Steuerungsebene die SPS. Diese steuert den Fertigungsprozess. Die 2. Stufe ist die Leitebene, diese leitet den Prozess eines einzelnen Fertigungsprozesses. Die Betriebsleitebene leitet schon die gesamte Fertigung. Die höchste Stufe der Automatisierungspyramide ist die Unternehmensebene, hier wird die Anbindung an das ERP-System eingeordnet.

Das Datenlogging kann zwischen Level 2 und 3 eingeordnet werden, da Daten zum Prozess gespeichert werden, diese aber auch für die gesamte Fertigung interessant sein könnten. Daher lag die Verantwortung dieses Projekts bei RENA bei dem Software-Team „Visualisierung“, welches für die Level 2 und 3 zuständig sind.

Bisher wurden die Daten mithilfe der eigenen Visualisierungs-Software und eines eingekauften Tools namens „ACCON-EasyLog“ von der Firma DELTA LOGIC [1] erfasst. Beides sollen nun überarbeitet oder ersetzt werden.

Für die Aufgabenstellungen wurden Lasten- / Pflichtenhefte (Abbildung 2 und Abbildung 3 sowie Abbildung 4) angelegt. Durch die stichpunktartige Aufführung der Lasten kann man die Gesamtaufgabe gut in einzelne Kleinaufgaben aufteilen. Dadurch wird der Arbeitsablauf besser strukturiert.

Im nachfolgenden werden die Aufgabenstellungen für die verschiedenen Tools aufgeführt.

TCP-Logging

Das TCP-Logging wird bisher von der RENA-Visualisierungs-Software der Sondermaschinen¹ übernommen. Da Serienmaschinen nicht mit dieser Visualisierungs-Software ausgestattet sind, verfügen sie bisher auch über kein TCP-Logging. Des Weiteren kann die Software auch ausgeschaltet werden, was zu einem Datenverlust führen würde.

Aktuell sind TCP-Telegramme definiert, welche die SPS an die Visualisierungs-Software sendet. Diese Telegramme müssen auch im neuen Daten-Logging unterstützt werden. Da das bisherige Logging als ein WPF-Projekt entwickelt wurde, können Teile des Codes

¹ Einzelmaschine Maschine welche für spezielle Produkte- und Produktionsanforderungen entwickelt und aufgebaut wird.

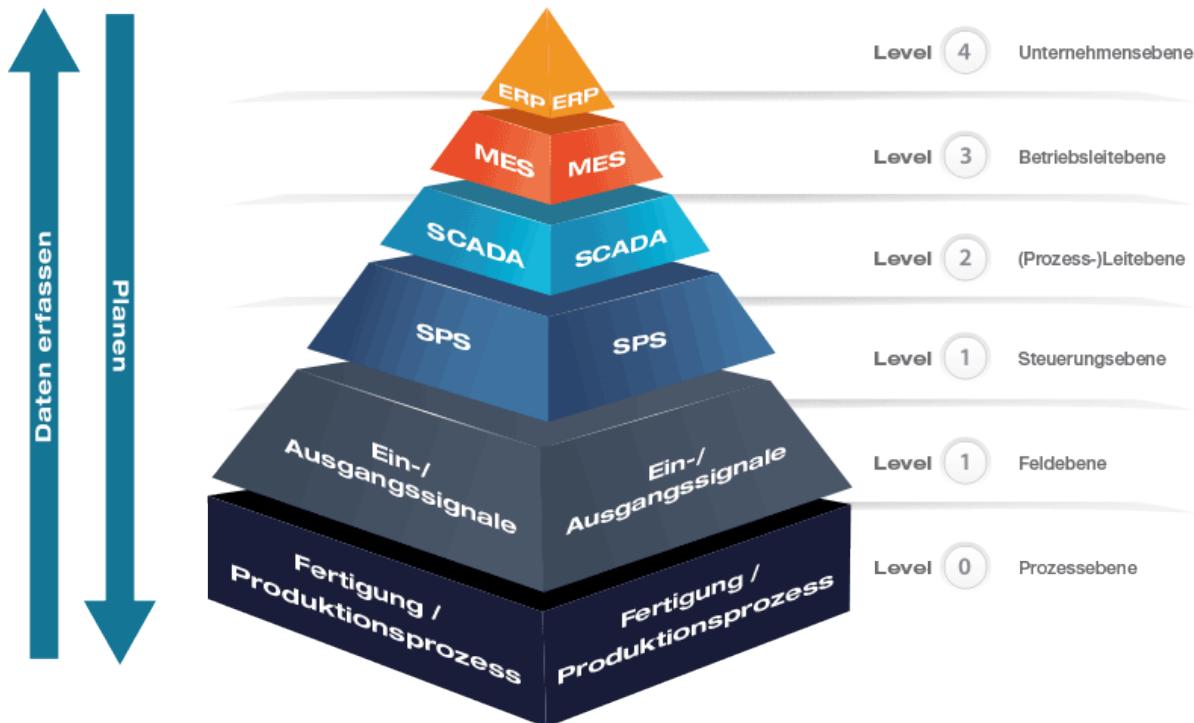


Abbildung 1: Automatisierungspyramide [2]

übernommen werden.

Der erste Schritt ist das Ausgliedern des Loggings aus der Visualisierungs-Software. Dies beinhaltet auch Verbesserungen und Anpassungen des aktuellen Codes. Die aktuelle Software erhält Telegramme per TCP-Kommunikation, welche dann in der Software ausgewertet und in die PostgreSQL-Datenbank gespeichert werden.

Da die Sondermaschine „RENA EPM 2 100F“ (Abbildung 5), mit dem TCP-Logging-Service ausgestattet wird, müssen noch zusätzliche, maschinenspezifische Telegramme integriert werden.

Beim Starten des TCP-Logging soll überprüft werden, ob die benötigten Tabellen existieren. Sollte dies nicht der Fall sein, so sollen diese automatisch erstellt werden. Diese Funktion soll über ein Ini-File aus- bzw. einschaltbar gemacht werden.

RENA CSV Service

Als zusätzliches „Kleinprojekt“ wurde das „RENA CSV Report“-Tool während der Entwicklung des TCP-Logging hinzugefügt. Da die „RENA EPM 2 100F“ (Abbildung 5) nicht über eine Visualisierungs-Software verfügt, sondern nur ein kleines Visualisierungs-Panel benutzt, können auch keine Reports in der Visualisierung erstellt werden. Daher wird ein unabhängiges Tool erstellt, welches einen Report über einen gesamten Produktionsprozess erstellt.

Die Reports sollen als CSV-File auf dem PC abgelegt werden. Diese Reports können in den nächsten 30 Tagen per USB-Stick „heruntergeladen“ werden. Danach sollen die Re-

R|E|N|A|

Lasten-/Pflichtenheft

	Beschreibung	TCP Logging
	Projektnummer	220412-25-001

Nr.	Kategorie	Lasten	Pflichten	Ansprechpartner	Bemerkung	Priorität	Status
1	Allgemein	Dienst im Hintergrund	Windows Service verwenden		mit TopSheet, damit man auch Debuggen kann	hoch	erledigt
2	Allgemein	Installations-Oberfläche	mit NSI eine Installations-EXE erstellen			gering	erledigt
3	Allgemein	einfache Fehlerdiagnose	über Logdaten			hoch	erledigt
1	Telegramm	Bisherige Telegramme integrieren				hoch	erledigt
2	Telegramm	Telegramm für Run EPM einführen	Neues Telegramm integrieren			hoch	erledigt
3	Telegramm	Telegramm für Alarm-Logging SPS einführen	Neues Telegramm integrieren		bisher über WinCC abgedeckt	hoch	erledigt
4	Telegramm	Telegramm für Process-Value-Logging einführen	Neues Telegramm integrieren			hoch	erledigt
5	Telegramm	Einfache Telegramm-Erweiterung	Dokumentation für Telegramm-Erweiterung schreiben			hoch	erledigt
6	Telegramm	Gültigkeitsprüfung der empfangenen Telegramme	Telegramm erst nach Überprüfung in Datenbank schreiben			hoch	erledigt
7	Telegramm	Siemens-WSTRING verarbeiten	Umwandlung Siemens-WSTRING zu string in C#			hoch	erledigt
1	Datenbank	Vorhandene Datenbank verwenden	Npgsql benutzen für PostgreSQL			hoch	erledigt
2	Datenbank	Datensicherheit	Daten puffern, falls diese nicht gespeichert werden können			gering	erledigt
3	Datenbank	Automatisch Tabellen erstellen und überprüfen	Nach dem ersten Starten des Service			hoch	erledigt

Abbildung 2: Lasten-/Pflichtenheft für „TCP Logging“

R|E|N|A|

Lasten-/Pflichtenheft

Beschreibung	CSV Service
Projektnummer	220412-25-003

Nr.	Kategorie	Lasten	Pflichten	Ansprech-partner	Bemerkung	Priorität	Status
1	CSV-Reports	Dienst im Hintergrund	Windows Service verwenden		mit Topshelf, damit man auch Debuggen kann	hoch	erledigt
2	CSV-Reports	Nach Produktionsende Report erstellen	Report generieren nach Benachrichtigung		Datenbank mit Benachrichtigung	hoch	erledigt
3	CSV-Reports	Report in ausgewählter Sprache	Initielle mit Sprachauswahl (LanguageID)		aktuell nur Englisch (US) und Deutsch unterstützt	hoch	erledigt
4	CSV-Reports	Datum in ausgewählter kultureller Darstellung	LanguageID für Kultur verwenden		aktuell nur Englisch (US) und Deutsch unterstützt	hoch	erledigt
5	CSV-Reports	Sortierbar über Prozessschritt	Jeder Datensatz mit Prozessschritt in Report		mittel	mittel	erledigt

Abbildung 3: Lasten-/Pflichtenheft für „CSV Service“

Lasten-/Pflichtenheft

	Beschreibung	SPS Logging
	Projektnummer	220412-25-002

R|E|N|A|

Nr.	Kategorie	Lasten	Pflichten	Ansprech-partner	Bemerkung	Priorität	Status
1	Allgemein	Dienst im Hintergrund	Windows Service verwenden		mit TopShelf, damit man auch Debuggen kann	hoch	erledigt
2	Allgemein	Installations-Oberfläche	mit NSIS eine Installations-EXE erstellen			gering	erledigt
3	Allgemein	einfache Fehlerdiagnose	Logging für Service implementieren			hoch	erledigt
4	Allgemein	Service steuerbar von "außen"	TCP-Kommunikation integrieren			mittel	erledigt
1	Datenbank	Automatisch Tabellen erstellen und überprüfen	Nach dem Laden der Konfiguration durchführen			hoch	erledigt
2	Datenbank	Vorhandene Datenbank verwenden	Npgsql benutzen für PostgreSQL			hoch	erledigt
3	Datenbank	Logging für jeden Datensatz ermöglichen	Symbolischer Zugriff ermöglicht dies		über MachineConfig-Datenbank	hoch	erledigt
4	Datenbank	Daten in Datenbank sortiert halten	In Modul-Tabellen aufteilen		über MachineConfig-Datenbank	hoch	erledigt
5	Datenbank	Erweiterung um weitere Variablen	Tabellen erweiterbar gestalten			hoch	erledigt
6	Datenbank	Tabellen auf überprüfen	Tabellen nach einlesen der MachineConfig überprüfen, ob Spaltenname und Datentyp noch übereinstimmen			hoch	erledigt
1	SPS	möglichst viele Datentypen unterstützen	Alle Datentypen welche AGLink lesen kann unterstützen			hoch	erledigt
2	SPS	Optimierte Bausteine unterstützen	Verwendung von AGLink von Delta Logic		Gleicher Hersteller wie beim ReSolve	hoch	erledigt
1	Konfiguration	Datenpunkte aus Access-Datenbank	Datenpunkte auslesen und verwenden			mittel	erledigt
2	Konfiguration	Flexibel für verschiedene Maschinen	Datenpunkte auslesen und verwenden		über MachineConfig-Datenbank	hoch	erledigt
3	Konfiguration	Datenbankzugriff konfigurierbar	Tabellen automatisch erstellen		Ini-File	hoch	erledigt
4	Konfiguration	Lese-Interval konfigurierbar	in Settings.ini		Ini-File	hoch	erledigt

Abbildung 4: Lasten-/Pflichtenheft für „SPS-logging“



Abbildung 5: RENA-Maschine „RENA EPM 2 100F“

porte gelöscht werden, um eine volle Festplatte zu vermeiden.

Die „RENA EPM 2 100F“ kann verschiedene Rezepte mit maximal 20 Rezeptschritten speichern. Jeder Datensatz soll einem Rezeptschritt zuordenbar sein. Durch das Datum und die Uhrzeit können auch Verläufe von Prozesswerten ermittelt werden.

Ein Report soll direkt nach dem Beenden den Prozesses erstellt werden.

SPS-Logging

Das SPS-Logging wird bisher von einem externen Tool „Accon-EasyLog“ durchgeführt. Dieses Tool soll nun durch ein selbst entwickeltes Programm ersetzt werden, da das Tool aufwändig zu konfigurieren ist. Zusätzlich fallen für jede mit SPS-Logging ausgestattete Maschine bisher Runtime-Lizenzkosten an.

Durch die Ersetzung des Tools durch ein eigenes Programm erhofft sich RENA, die Entwicklungszeiten und Maschinenkosten senken zu können.

Im ersten Schritt soll eine Kommunikationsbibliothek gefunden werden, welche in das .NET-Framework integrierbar ist. Diese Bibliothek muss darüber hinaus symbolisch auf die Variablen in der SIEMENS-SPS zugreifen können.

In eine „Machine-Config“-Datenbank werden die Informationen für das Programm zur Verfügung gestellt. Diese müssen eingelesen und verarbeitet werden. Diese Maschinenkonfiguration enthält für das Programm auch unwichtige Angaben, diese müssen herausgefiltert werden.

Um das Programm steuern zu können soll ein Kommunikationskanal per TCP/IP inte-

griert werden, welcher beispielsweise ein Neuladen der Log-Konfigurations-Daten veranlassen kann. Darüber hinaus soll auch ein ausgehender Kanal integriert werden. Über diesen Kanal könnten Alarm-Meldungen an andere Programme weitergegeben werden. Als letzter Schritt steht das Testen an. Hierfür kann ein Teststand, welcher für Entwicklungszwecke die Software einer RENA-Maschine simulieren kann, genutzt werden. Dort soll das Programm installiert werden und die geloggten Daten analysiert werden.

3 Konzept

3.1 Logging

SIEMENS SPSEN bieten die Möglichkeit, mit vorbereiteten Funktionsbausteinen Daten in eine CSV-Datei zu loggen. Diese CSV-Datei wird dann auf der Speicherplatte der SPS gespeichert und kann über einen Webserver, welcher auf der SPS läuft, heruntergeladen werden. Dies belastet den Speicherplatz auf der Speicherplatte und die CSV-Dateien müssten händisch oder mit einem extra Tool von der Speicherplatte auf einen PC heruntergeladen werden. Daher hat sich RENA dafür entschieden, das Logging auf einen PC auszulagern und nicht auf der SPS durchzuführen. Hinzu kommt die Datenmenge, welche in den CSV-Dateien gespeichert werden müsste. Die Vorgabe für den maximalen Speicherbedarf des Loggings beträgt etwa 50 Gigabyte. Eine solch große Speicherplatte bietet SIEMENS nicht an.

3.2 Anwendung

Bei RENA werden in den meisten Maschinen Industrie-PCs eingesetzt, welche mit Windows 10 als Betriebssystem ausgestattet sind. Auf diesen PCs soll die Anwendungen für das Datenlogging ausgeführt werden. Aktuell wird beispielsweise die Visualisierungs-Software auf diesen PCs ausgeführt. Durch diese Vorgabe steht das Betriebssystem, für welches die Anwendungen entwickelt werden, fest.

Windows kann unter anderem Desktop-Anwendungen, Services oder Aufgaben (engl. *Tasks*) ausführen.

Eine Desktop-Anwendung kann entweder vom Benutzer nach der Anmeldung gestartet werden oder die Anwendung kann direkt nach der Anmeldung automatisch gestartet werden. In diesem Fall werden Daten nur geloggt, wenn ein Benutzer am PC angemeldet ist, was zu einem Datenverlust führen kann. Darüber hinaus können Desktop-Anwendungen vom Benutzer geschlossen bzw. beendet werden. Dies ist nicht erwünscht und sollte nur von einem befähigten Benutzer durchgeführt werden (einem Administrator).

Eine Aufgabe kann im Windows-Aufgabenplaner angelegt werden. Die Aufgabe führt ein Skript oder eine EXE-Datei aus, auch wenn kein Benutzer angemeldet ist. Dadurch wäre das Problem des Datenverlustes, welches Desktop-Anwendungen hätten, gelöst. Allerdings kann eine Aufgabe höchstens einmal pro Minute aktiviert werden, was für Logging-Tools zu langsam ist.

Ein Windows-Service schließlich kann automatisch gestartet werden, ohne dass sich ein Benutzer anmeldet. Außerdem läuft ein Windows-Service dauerhaft im Hintergrund und kann nur durch einen Administrator gestoppt bzw. beendet werden.

Weiter soll das Datenlogging nicht an ein spezielles Visualisierungssystem gebunden sein, um auch in Zukunft entsprechenden Kundenwünschen, ein anderes Visualisierungssystem

einzusetzen, gerecht zu werden.

Diese Punkte führten zu der Entscheidung, dass die Anwendungen als Windows-Service entwickelt werden.

Der Vorschlag der Entwicklungsabteilung, die Anwendungen im .NET-Framework mit C# zu entwickeln, wurde umgesetzt. C# ist eine typsichere, objektorientierte Programmiersprache, welche komplett in das .NET-Framework integriert ist.

Allerdings könnten auch andere Programmiersprachen verwendet werden, beispielsweise Python. Mit dieser kann auch ein Windows-Service mit den entsprechenden Bibliotheken entwickelt werden. Allerdings ist dies bei RENA eher eine selten eingesetzte Programmiersprache.

3.3 Datenspeicherung

Um die Logging-Daten zu speichern, gibt es mehrere Möglichkeiten:

Zur Speicherung der Daten wird wie bisher eine Datenbank gewählt, da diese den Vorteil bietet, eine große Anzahl von Datensätzen performant zu lesen, zu schreiben und zu verarbeiten. Des Weiteren ist der Mehrbenutzer-Betrieb von Vorteil, da dadurch das Logging unabhängig von der Auswertung auf die Daten zugreifen kann. Würde man die Daten in Dateien speichern, wäre dies nicht ohne einen Mehraufwand möglich.

Bisher wurde eine PostgreSQL-Datenbank verwendet, diese Datenbank ist eine objektrelationale Datenbank. Es ist SQL-fähig und es gibt bereits Bibliotheken, mit welchen die Daten aus einem C#-Programm in die Datenbank sicher und schnell übertragen werden können. Des Weiteren können Daten flexibel repräsentiert werden und eine nachträgliche Veränderung ist einfach umzusetzen. Auch die erhöhte Übersichtlichkeit der Daten und der Struktur, im Gegensatz zu netzartigen oder hierarchischen Datenbanken, ist ein Vorteil der objektrelationalen Datenbanken.

Die Entscheidung für eine PostgreSQL-Datenbank wurde aus Kosten-, Lizenz-, Support und Performancegründen gewählt. Andere Datenbanksysteme wie z.B. MySQL oder Microsoft SQL kosten Lizenzgebühren, wenn man sie kommerziell nutzt. Die Visualisierungs-Software WinCC benutzt eine Microsoft SQL-Datenbank, um interne Daten zu speichern. Daher entstand die Idee, diese auch für das Datenlogging der RENA zu verwenden. Dies wäre allerdings nur möglich, wenn pro Maschine eine Lizenz von Microsoft gekauft wird. Um die Stabilität des bestehenden Visualisierungssystems weiter gewährleisten zu können und die Produktionskosten nicht zu erhöhen, wurde diese Idee nicht umgesetzt. Darüber hinaus hat PostgreSQL eine lebendige Community, welche das Datenbanksystem dauerhaft weiterentwickelt. Ebenfalls ist die PostgreSQL-Schnittstelle in AcconEasy-Log integriert, auch dies war ein weiterer Grund für PostgreSQL.

3.4 Projektplan

Bevor mit der Umsetzung des Projekts begonnen wird, wird zunächst für jede Teilaufgabe ein Projektplan (Abbildung 6, Abbildung 7 und Abbildung 8) erstellt. Vom Projektplan kann immer der aktuelle Projektstand abgelesen werden. Bei RENA wird für jedes Projekt ein Projektplan vom Projektmanagement erstellt. Für kleinere Software-Projekte werden die Projektpläne von den Entwicklern selbst erstellt.

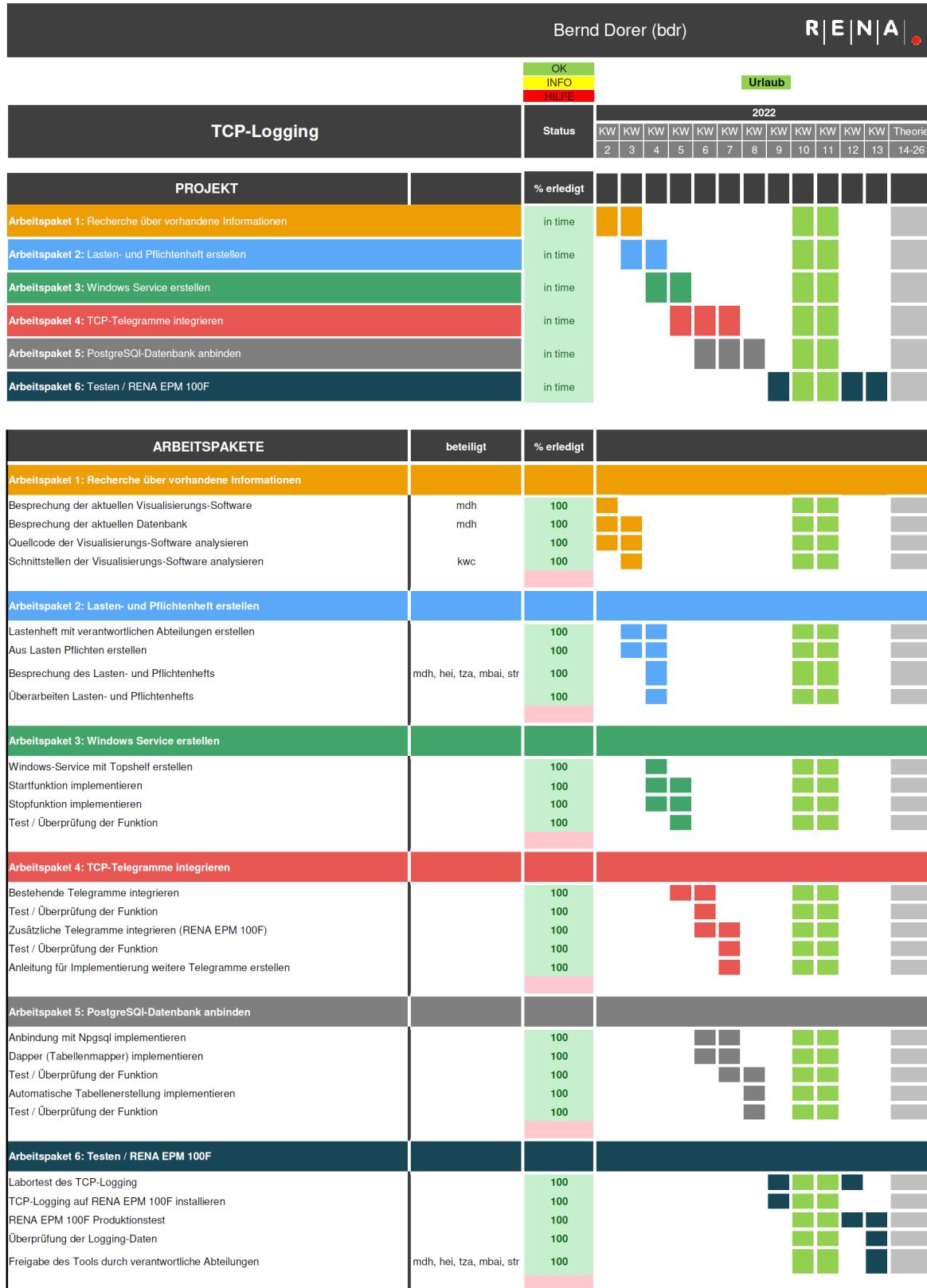


Abbildung 6: Projektplan TCP-Logging

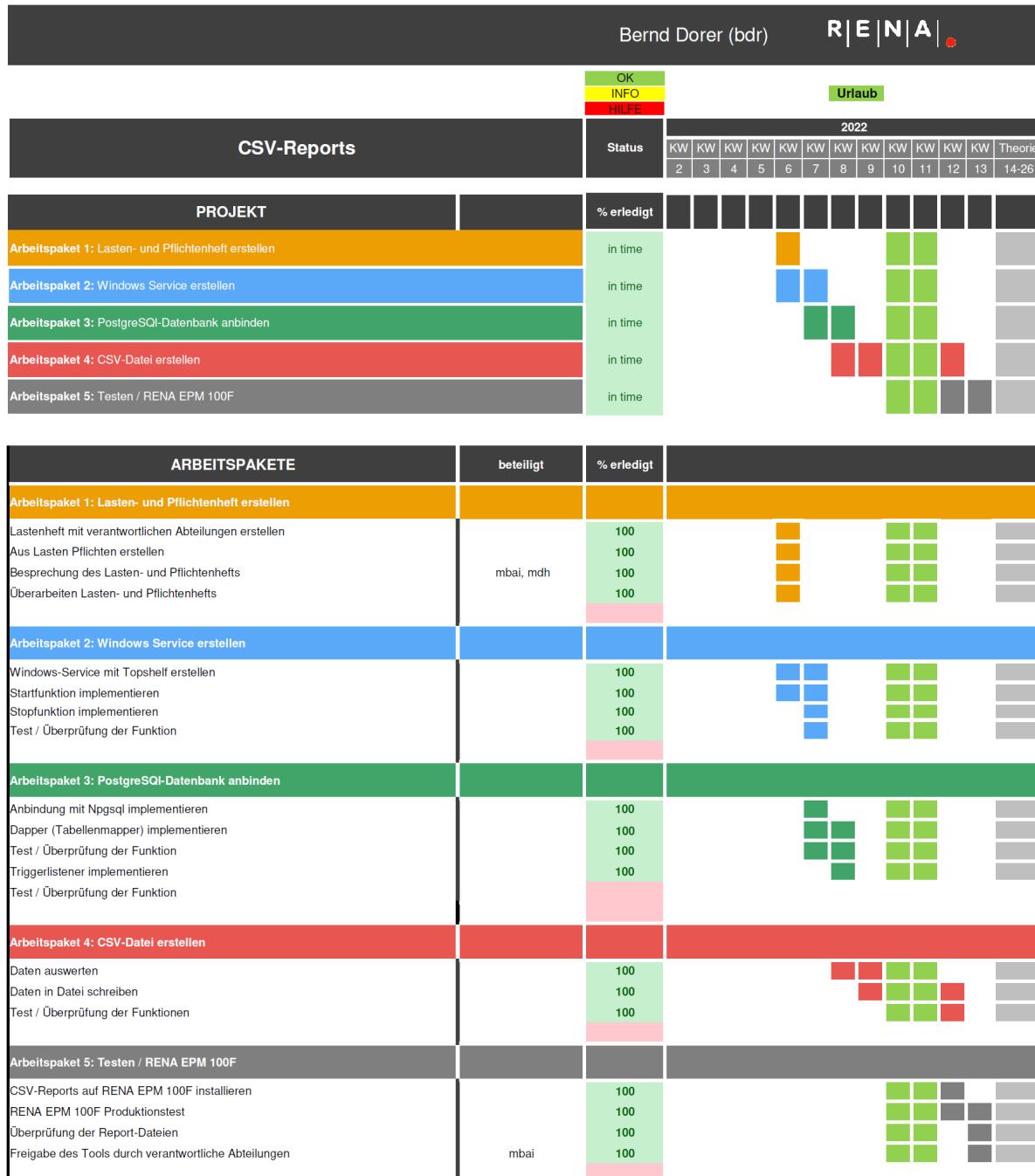


Abbildung 7: Projektplan CSV-Reports

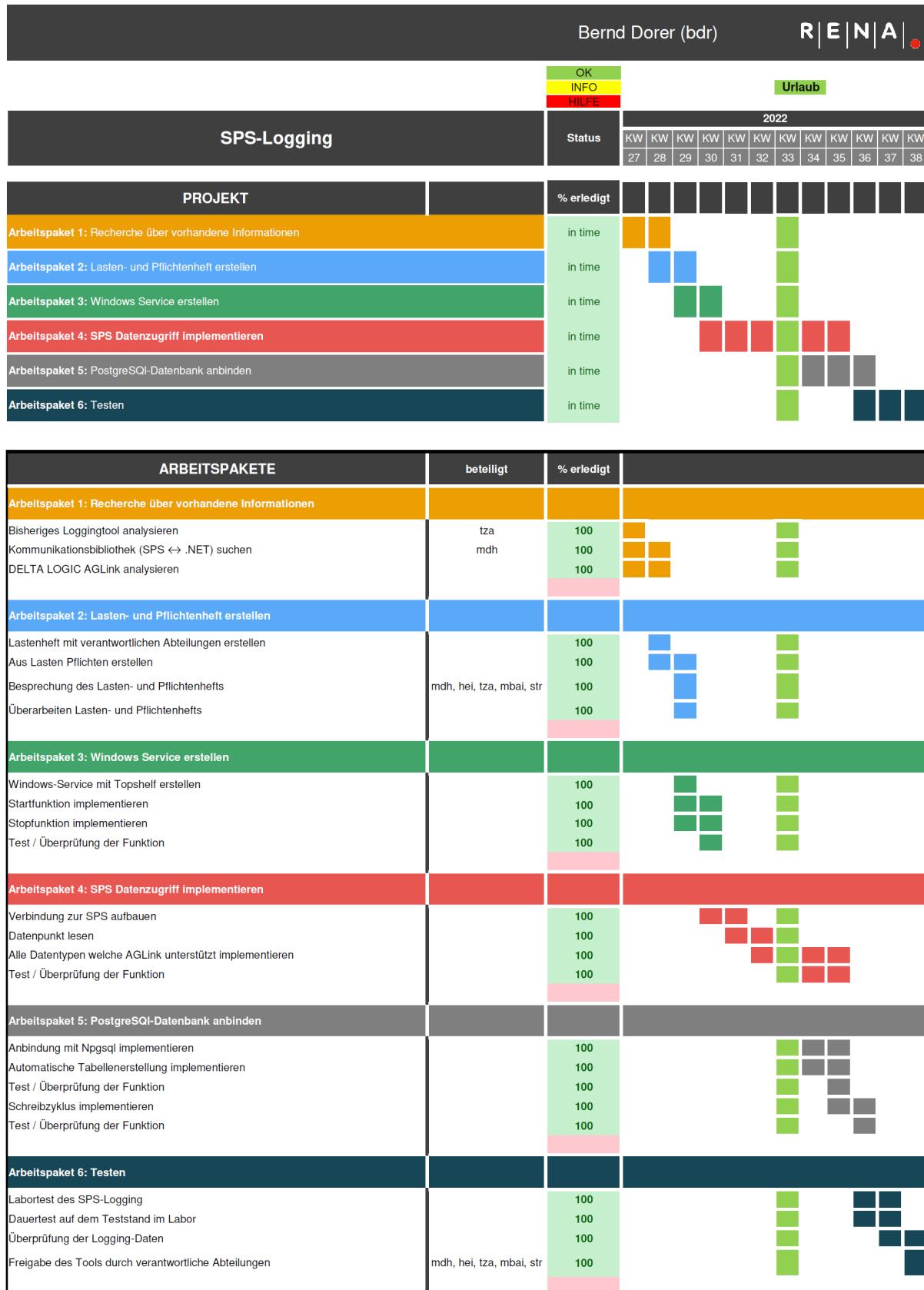


Abbildung 8: Projektplan SPS-Logging

4 Recherche

4.1 .NET

.NET [3] ist eine Sammlung verschiedener Software-Plattformen, welche zur Entwicklung und Ausführung von Programmen dient. Entwickelt und herausgegeben wurde .NET von Microsoft. Die Plattform wurde 2002 veröffentlicht und ist seit 2014 eine Open-Source-Software. Mit dem .NET-Framework lassen sich Webseiten, Dienste, Desktop-Anwendungen und vieles mehr entwickeln, allerdings nur für Windows. Mit .NET-Core, welches 2016 veröffentlicht wurde, wird der Code nativ auf jedem kompatiblen Betriebssystem ausgeführt. Dadurch schafft man eine vom Betriebssystem unabhängige Plattform. Als Hauptsprachen werden von .NET C#, Visual Basic und F# unterstützt, allerdings sind auch weitere Sprachen wie zum Beispiel Python, JavaScript oder C++ nutzbar. Bei RENA wie im .NET Bereich auf C# gesetzt.

Das Buch „Die .NET-Technologie“ [4] liefert ein Grundverständnis der .NET-Technologie.

4.2 WPF

Windows Presentation Foundation (WPF) [5] ist eine Benutzeroberfläche-Framework, mit welcher beispielsweise Desktop-Anwendungen entwickelt werden können. WPF gehört zu .NET und unterstützt viele Anwendungsentwicklungsfeatures. Wie zum Beispiel Ressourcen, Steuerelemente, Grafik und Layout. Bei RENA wird WPF zur Erweiterung der Visualisierungssoftware WinCC von SIEMENS verwendet. Durch WPF können spezielle Werkzeuge und Darstellungen entwickelt werden, welche genau auf den Kunden abgestimmt werden können. Bei RENA werden die WPFs beispielsweise für die Navigation eingesetzt (Abbildung 9). Eine solche Navigation wäre ausschließlich mit WinCC Funktionen sehr schwer umsetzbar und extrem unflexibel. Darüber hinaus sind die WPFs flexibler einsetzbar als die WinCC-Komponenten.

4.3 C#

„C# (Aussprache „C Sharp“) ist eine moderne, objektorientierte und typsichere Programmiersprache. C# ermöglicht Entwicklern das Erstellen zahlreicher sicherer und robuster Anwendungen, die in .NET ausgeführt werden.“ [6] so Microsoft auf ihrer Webseite. C# wird in eine Zwischensprache kompiliert, welche anschließend auf einem virtuellen System (Common Language Runtime) ausgeführt wird (ähnlich wie Java). Durch die vielen Bibliotheken (Libraries), welche meist kostenlos im NuGet-Manager installierbar sind, werden Programme häufig einfacher und sicherer, da der Code der Bibliotheken meist getestet wurde. Der NuGet-Manager enthält über 90.000 Packages und ist voll integriert in das Visual Studio von Microsoft.



Abbildung 9: Navigations-WPF einer RENA-Maschine

FC_Event_Alarm_Log [FC2908]
FC_Event_Check_Buffer [FC2900]
FC_Event_Dosage_Log [FC2904]
FC_Event_Module_Log [FC2903]
FC_Event_Run_Log [FC2902]
FC_Event_Value_Log [FC2906]
FC_Get_stRecipeStepData [FC2907]
FC_Get_wsHeader [FC2909]
FC_LogLoggingData [FC2910]
FC_MOD_TCPIPLogging_CALL [FC900]
FC_MOD_TCPIPLogging_CONF [FC901]
FB_MOD_TCPIPLogging [FB900]

Abbildung 10: Funktionen und Funktions-Baustein aus dem TIA-Portal, welche zum Logging verwendet werden.

Das C# Kompendium [7] wird als Nachschlagewerk verwendet, durch zahlreiche Codebeispiele und Beschreibungen werden auch komplexere Aufbauten verständlich.

4.4 Telegramme

Die bestehenden Telegramme können teilweise aus dem Quellcode des bestehenden Data-Loggings sowie auch von der SPS-Software (Abbildung 10) entnommen werden. Die Funktionen (bspw. „FC_Event_Batch_Log“) generieren den zu sendenden String, welcher mit dem Funktionsbaustein „FB_TCP_Send_Log“ gesendet wird. Für folgende Projekte oder eventuelle Erweiterungen wurde ein Dokument (Abbildung 14) mit der Beschreibung aller Telegramme erstellt.

Beispiele für Telegramme ist das „Consumption“-Logging oder das Prozessdaten-Logging. Um Verbrauchsdaten zu erfassen, wurde das „Consumption“-Logging-Telegramm (Abbildung 11) eingeführt. Des Weiteren wurden neue Telegramme speziell für die „RENA EPM 2“ hinzugefügt (Unter anderem das ProcessValue-Telegramm (Abbildung 12) welches keine feste Anzahl an Daten überträgt). Es ist erweiterbar um jeweils eine zugehörige Wert-ID und den Wert selbst. Dieses Telegramm kann natürlich auch in Zukunft von anderen Maschinen verwendet werden, nicht nur von der „RENA EPM 2“.

Telegram				
EventType	Index	TimeStamp	Chemistry	Consumption_Value
int	int	string	int	real
50				

Abbildung 11: Telegramm des Consumption-Loggings

Telegram										
EventType	Index	TimeStamp	moduleid	componentid	typeid	value1id	value1	value2id	value2	...
int	int	string	int	int	int	long int	real	long int	real	...
160										

Abbildung 12: Telegramm des Process-Value-Loggings

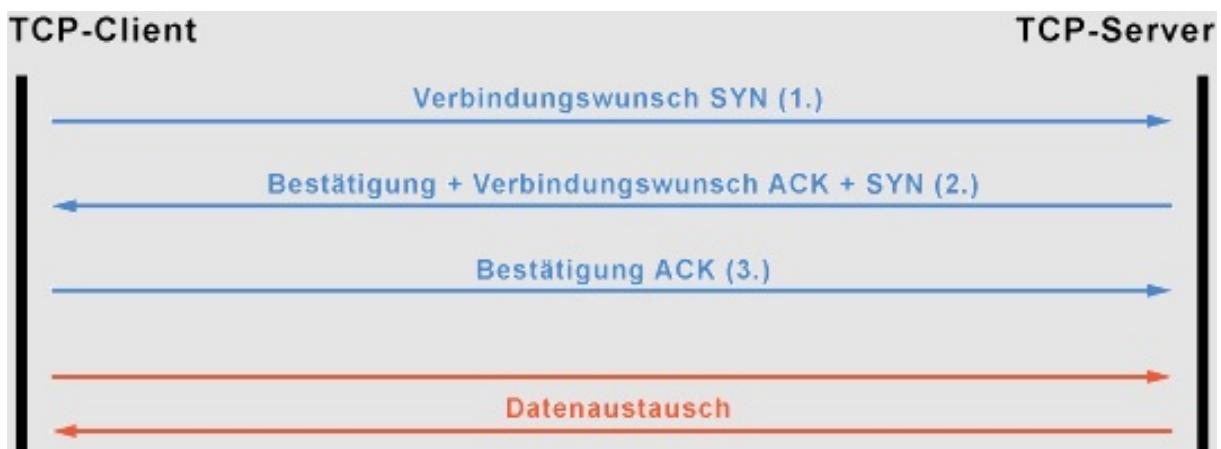


Abbildung 13: Verbindungsauflaufbau einer TCP-Kommunikation [8]



Abbildung 14: Alle unterstützten Telegramme und Datenbank-Tabellen

4.5 TCP

TCP steht für *Transmission Control Protocol* und ist ein Netzwerkprotokoll. Eine Verbindung besteht aus zwei Endpunkten, sogenannten Sockets, welche in beide Richtungen kommunizieren können. Der Verbindungsauflaufbau (Abbildung 13) ist ebenfalls fest definiert, ebenso wie auch der Datenaustausch. TCP ist ein sicheres Übertragungsprotokoll, da auf jedes empfangene Paket mit einem Bestätigungs-Paket (Acknowledge-Paket) geantwortet wird. Sollte der Sender keine Antwort erhalten, so wird das Paket erneut gesendet bzw. nach mehreren Fehlversuchen abgebrochen. Die IETF (Internet Engineering Task Force) hat die TCP-Kommunikation in dem Standarddokument RFC793 [9] definiert. Die IETF ist ein Arbeitsgruppe einer internationalen Community; welche Standardbetriebsprotokolle definiert, zu diesen Protokollen zählt auch das TCP-Protokoll.

4.6 PostgreSQL

Als Datenbank wurde beim bestehenden Logging eine PostgreSQL-Datenbank [10] verwendet, welche auch wieder verwendet wird. PostgreSQL ist eine objektrelationale Datenbank, welche das Bindeglied zwischen relationalen und objektorientierten Datenbanken darstellt. Die Datenbank erschien schon 1996 und wird seit 1997 von einer Open-Source-Community weiterentwickelt. PostgreSQL lehnt sich stark an den SQL-Standard an, allerdings gibt es auch ein paar PostgreSQL-spezifische Funktionen.

Durch die bestehende Datenbank konnten die Tabellen, welche für das TCP-Logging verwendet werden, übernommen werden. Für das IO-Logging wird ebenfalls der Tabellenentwurf der bereits bestehenden Tabellen übernommen, damit die bestehende Auswertung nicht angepasst werden muss.

4.7 Kommunikationsbibliothek (.NET ↔ SIEMENS-SPS)

Für die Kommunikation zwischen dem Service und der SIEMENS-SPS wird eine Kommunikationsbibliothek benötigt. Bisher wird bei RENA die AGLink-Bibliothek von DELTA LOGIC verwendet, allerdings kann diese Version kein symbolischen Zugriff auf Variablen ausführen. Der symbolische Zugriff wird benötigt, da bei optimierten Datenbausteinen¹ von SIEMENS nicht wie bisher über eine direkte Adressierung zugegriffen werden kann. Nach mehreren Anfragen bei verschiedenen Software-Herstellern wurde festgestellt, dass nur DELTA LOGIC mit einer anderen Ausführung der AGLink-Bibliothek den symbolischen Zugriff durchführen kann. Daher wird diese Version eingesetzt. Die mitgelieferte API-Dokumentation (Abbildung 15) enthält Informationen zu den einzelnen Funktionen

¹Speicher und Zugriffs optimierte Bausteine, welche für die Datenspeicherung zuständig sind auf

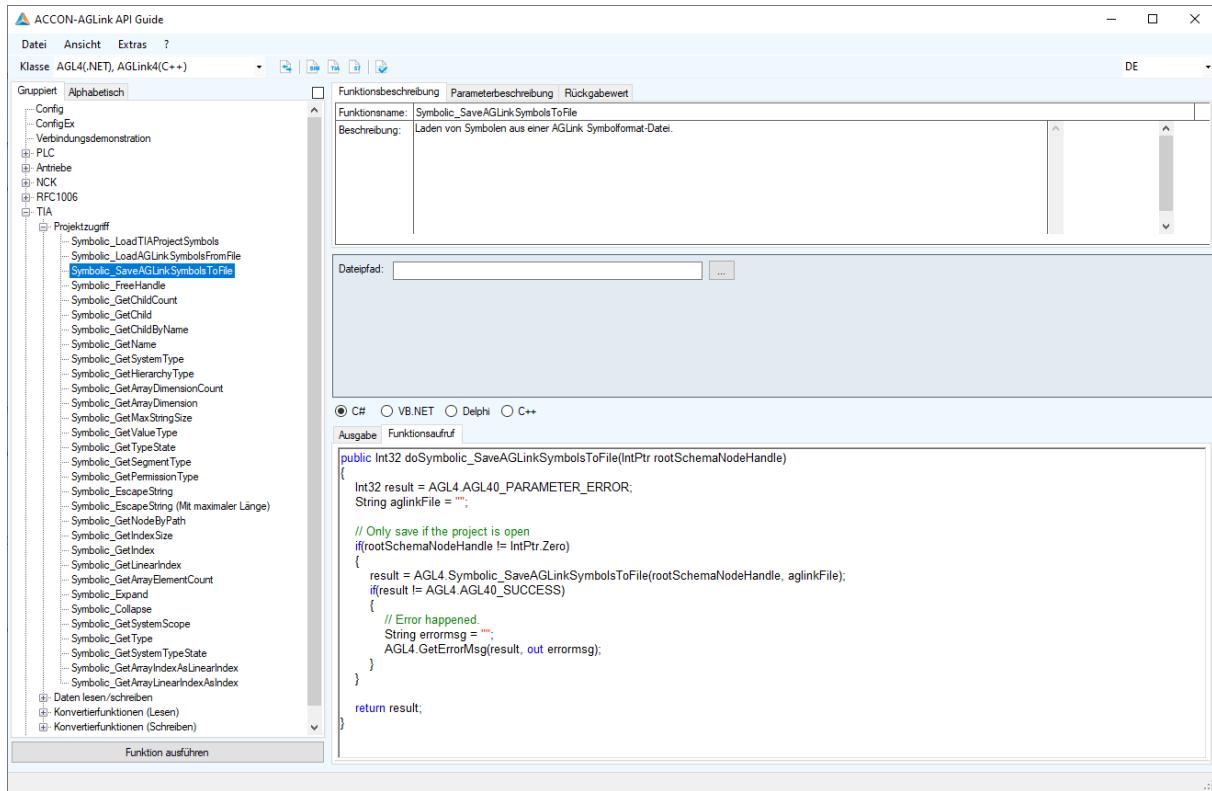


Abbildung 15: API-Guide für AGLink Bibliothek

und auch Beispiel-Codeausschnitte.

5 TCP-Logging

Der Service wartet auf Daten, welche per TCP übertragen werden. Sobald der Service Daten empfängt, wandelt er diese und speichert die Daten in der PostgreSQL-Datenbank. Der grobe Ablauf ist im Programmablaufplan (Abbildung 16) zusehen.

Da das TCP-Logging bisher in der Visualisierungs-Software enthalten ist, konnten einige Programmteile übernommen werden.

Eingesetzt wird der TCP-Logging-Service bisher hauptsächlich auf Sondermaschinen, allerdings ist eine Ausweitung des Projekts geplant.

5.1 TCP-Kommunikation

Für die Kommunikation wurde ein „Ethernet-Controller“ integriert, der vom bestehenden Projekt übernommen und dessen Code nur leicht verändert wurde. Dieser Controller wird als Server verwendet, welcher dauerhaft auf Nachrichten von Clients wartet. Mithilfe eines Sockets kann ein solcher Server realisiert werden. Dieser Socket ist an eine Port-Nummer gebunden, so dass die Transport-Schicht (Abbildung 17) die Anwendungen aufteilen kann. Das OSI-Model beschreibt die verschiedenen Netzwerkschichten nach ISO 7498-1:1994 [11].

Die TCP-Kommunikation wird vom bisherigen Daten-Logging übernommen und nur leicht angepasst.

Die Klasse `EnternetController` nutzt Sockets von der Klasse „`System.Net.Sockets`“, welche für die Kommunikation zuständig sind.

```

1 private void WaitClient()
2 {
3     var permission = new SocketPermission(NetworkAccess.Accept, TransportType.Tcp,
4         string.Empty, SocketPermission.AllPorts);
5     _server = null;
6     permission.Demand();
7     _server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
8     _server.Bind(new IPEndPoint(IPAddress.Any, _port));
9     _server.Listen(BACKLOCK);
10    WaitAccept();
}

```

Codelisting 1: Initialisierung des Sockets (C#)

Beim Initialisieren des Sockets (Codelisting 1) wird ein neuer Socket erstellt (Zeile 6). Diesem wird als Übergabeparameter die Adressfamilie mitgeteilt, die InterNetwork-Familie ist die IPv4 Adress-Familie. Als Socket-Typ wurde `Stream` gewählt und das verwendete Protokoll ist TCP, welches ebenfalls als Übergabe-Parameter an den Socket

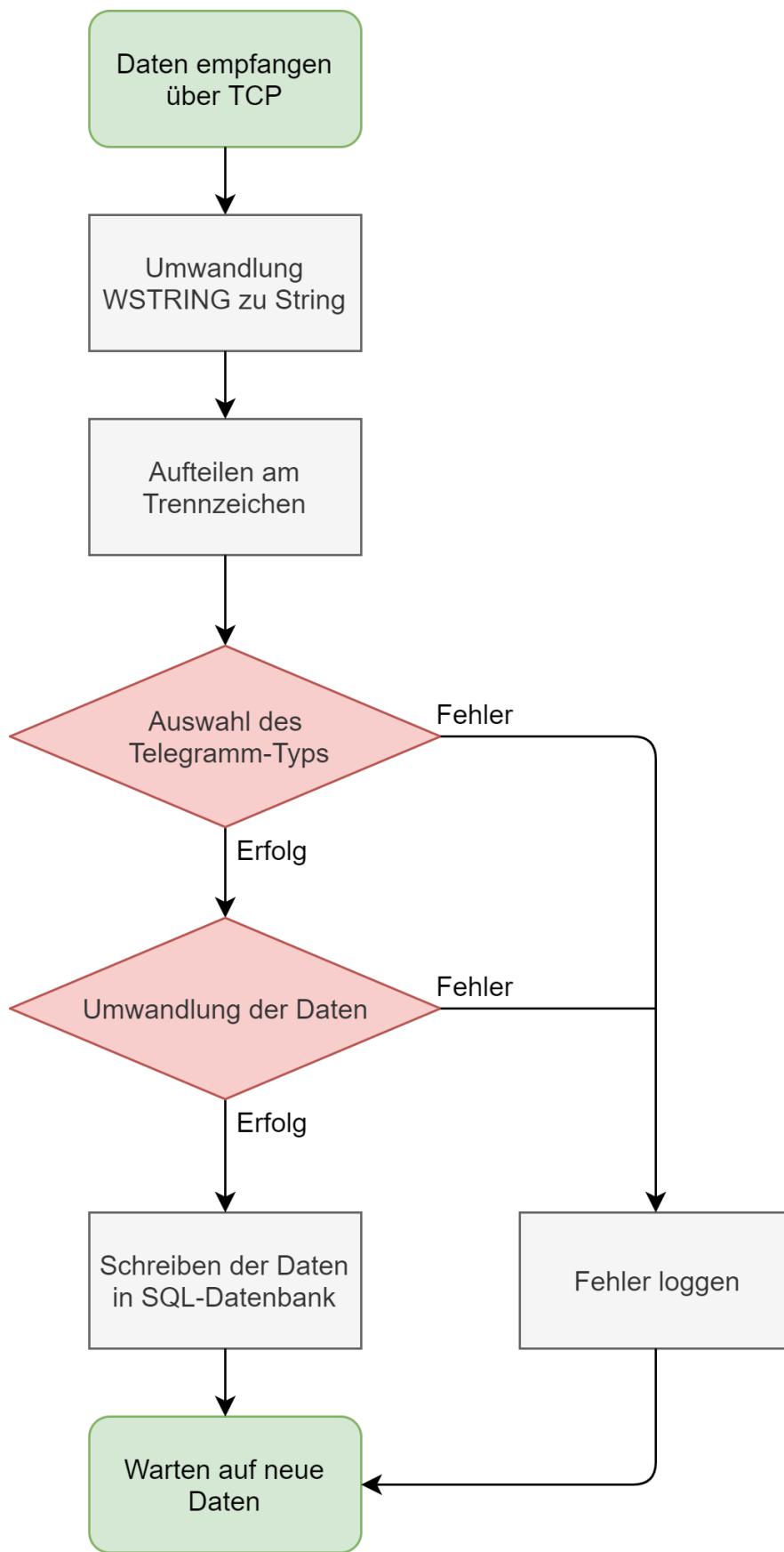


Abbildung 16: Programmablaufplan TCP-Logging

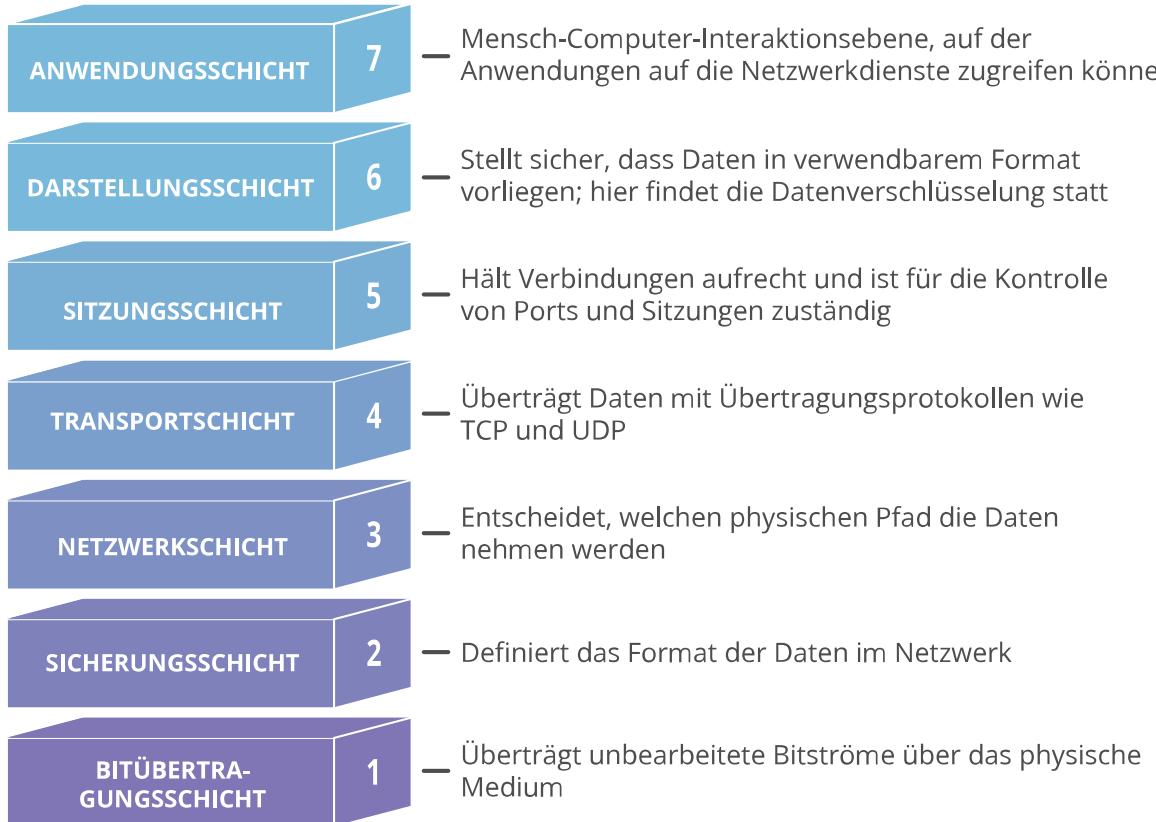


Abbildung 17: OSI-Modell für Netzwerkprotokolle als Schichtenarchitektur [12]

übergeben wird. Anschließend wird in Zeile 7 ein neuer Endpunkt einer Kommunikation angelegt, dieser Endpunkt kann Nachrichten von allen IP-Adressen am definierten Port empfangen. In der Zeile 8 wird die Anzahl der gepufferte Anfragen festgelegt (Backlog).

```

1 private void WaitAccept()
2 {
3     var listenerCallback = new AsyncCallback(AcceptCallback);
4     _server.BeginAccept(listenerCallback, _server);
5 }
```

Codelisting 2: Callback auf Socket starten (C#)

Danach wird die Funktion `WaitAccept()` (Codelisting 2) aufgerufen. Diese Funktion definiert eine asynchrone Callback-Methode, welche den eingehenden Verbindungsaufbau annimmt und für die Kommunikation zum anfragenden Socket einen neuen Socket erstellt. Dieser Socket startet dann das Empfangen der Daten.

```

1  private void ReceiveCallback(IAsyncResult data)
2  {
3      try
4      {
5          var container = data.AsyncState as object[];
6          var buffer = container[0] as byte[];
7          _handler = container[1] as Socket;
8          var receiveBufferLength = _handler.EndReceive(data);
9
10         if ((receiveBufferLength > 0) && !_disposing)
11         {
12             //_receiveBuffer.Enqueue(Encoding.ASCII.GetString(buffer, 0,
13             //receiveBufferLength));
14             var receivedBytes = new byte[receiveBufferLength];
15             Array.Copy(buffer, receivedBytes, receiveBufferLength);
16             _receiveBufferQueue.Enqueue(receivedBytes);
17             var receiveBuffer = new byte[BUFFER_SIZE];
18             container = new object[2];
19             container[0] = receiveBuffer;
20             container[1] = _handler;
21             _handler.BeginReceive(receiveBuffer, 0, receiveBuffer.Length,
22                 SocketFlags.Partial, new AsyncCallback(ReceiveCallback), container);
23         }
24         else
25         {
26             _handler.Close();
27             OnStatusChanged(_handler.Connected, false);
28         }
29     }
30     catch (Exception ex)
31     {
32         ErrorStatus(_handler.Connected, ex);
33     }
34 }
```

Codelisting 3: Callback, wenn TCP-Daten empfangen wurden (C#)

Die Funktion `ReceiveCallback()` (Codelisting 3) empfängt asynchron Daten, welche dem Empfangsbuffer (`_receiveBufferQueue`) angefügt werden (Zeile 15). Zu Beginn jedoch werden die empfangenen Daten asynchron geladen (Zeile 5 bis 8) und es wird überprüft, ob überhaupt Daten empfangen wurden. Sollten Daten empfangen worden sein, werden diese zwischengespeichert (Zeile 13 und 14) und anschließend dem Empfangsbuffer angehängt (Zeile 15). Anschließend wird der Callback erneut gestartet, bis keine Daten mehr empfangen werden, oder der Ethernet-Controller beendet werden soll, dann wird der Socket geschlossen (Zeile 24).

```

1  private void RunReceive()
2  {
3      var buffer = new List<byte>();
4      while (true)
5      {
6          while (!_receiveBufferQueue.IsEmpty)
7          {
8              if (_receiveBufferQueue.TryDequeue(out var raw))
9              {
10                  buffer.AddRange(raw);
11              }
12          }
13          if (buffer.Count > 0)
14          {
15              DataReceived?.Invoke(this, new ReceiveEventArgs(true, buffer.ToArray()));
16              buffer.Clear();
17          }
18          Thread.Sleep(RECEIVE_DELAY);
19      }
20 }

```

Codelisting 4: RunReceive-Thread (C#)

In dem Thread `RunReceive` (Codelisting 4) werden die Daten in einen Event-Handler übergeben. Der Event-Handler übergibt die Nachricht und die Daten an einen anderen Thread, somit ist das Empfangen und Verarbeiten der Daten in unterschiedliche, von einander unabhängige Threads aufgeteilt. Sie sind unabhängig voneinander. Da immer neue Daten im Puffer stehen können, wird in einer Dauerschleife (Zeile 4) der Buffer überprüft. Zur Entlastung des Systems wird der Thread für eine definierte Zeit (hier 10ms) pausiert (Zeile 18). Solange der Empfangs-Puffer nicht leer ist (Zeile 6), werden die Bytes in einen anderen Puffer kopiert (Zeile 10) und aus dem Receive-Buffer entfernt (Zeile 8). Nachdem der Empfangs-Puffer kopiert und geleert wurde, wird überprüft, ob Daten im Puffer stehen (Zeile 13). Wenn ja, dann werden diese dem Event-Handler „`DataReceived`“ hinzugefügt (Zeile 15) und anschließend der Buffer wieder geleert (Zeile 16).

5.2 Datenkonvertierung

Die Daten, welche von der SPS gesendet werden, sind in der SPS als „WSTRING“ abgelegt. Der WSTRING wurde von SIEMENS hinzugefügt, um Unicode-Strings zu unterstützen und ein Zeichen (WCHAR) ist 16 Bit groß. Außerdem reserviert der Compiler von SIEMENS einen Speicherbereich vorgegebener Länge. In den ersten 16 Bits werden für die Kapazität des WSTRINGS verwendet, die nächsten 16 Bits für die tatsächlich verwendete Länge des WSTRINGS, anschließend folgend die Zeichen.

```

1  /// <summary>
2  /// Get Message Strings from received bytes
3  /// </summary>
4  /// <param name="receivedBytes"></param>
5  private void HandlePlcLoggingMessageReceived(byte[] receivedBytes)
6  {
7      var allBytes = receivedBytes;                      //is WSTRING from PLC ==> one or more
8      telegrams transmitted by TCP
9      var allBytesCount = allBytes.Length;               //total length of received byte stream by
10     tcp
11     var processedBytesCount = 0;                      //count of already processed bytes
12     (converted to message information)
13
14     while (processedBytesCount < allBytesCount)
15     {
16         var totalLengthOfMessage = 2 *
17             BitConverter.ToInt16(allBytes.GetBytes(processedBytesCount,
18             2).Reverse().ToArray(), 0);
19         var actualLengthOfMessage = 2 *
20             BitConverter.ToInt16(allBytes.GetBytes(processedBytesCount + 2,
21             2).Reverse().ToArray(), 0);
22         var messageBytes = allBytes.GetBytes(processedBytesCount + 4,
23             actualLengthOfMessage).ToArray();
24         var messageString = Encoding.BigEndianUnicode.GetString(messageBytes, 0,
25             messageBytes.Length);
26         //_logger.Debug($"PLC Logging TCP/IP Server: DataReceived (message):
27         // {messageString}");
28         var messageSplittedStrings = messageString.Split(MessageStringDelimiter);
29
30         //Handle the message if valid message
31         if (!string.IsNullOrWhiteSpace(messageString))
32         {
33             //Add Data to ReceivedDataHandler.
34             ReceivedDataHandler?.Invoke(this, messageSplittedStrings);
35         }
36         processedBytesCount += totalLengthOfMessage + 4;
37     }
38 }
```

Codelisting 5: Daten-Konvertierung Bytes zu String (C#)

In Zeile 13 und 14 des Codelisting 5 werden die Kapazität und die verwendete Länge berechnet und anschließend mit 2 multipliziert, da ein WCHAR doppelt so viele Bits hat wie ein Byte. Anschließend werden in Zeile 15 die verwendeten Bytes aus dem Byte-Array herauskopiert, damit diese von der Klasse `Encoding` in einen String konvertiert werden kann, was in der Zeile 16 ausgeführt wird. In Zeile 18 werden die Daten anschließend mit der Split-Methode am Trennzeichen aufgetrennt. Nach der Überprüfung des empfangenen Strings, ob dieser NULL oder ein leerer String ist, wird ein Event dem `ReceivedDataHandler` (mit dem String-Array der empfangenen Daten) hinzugefügt. Danach wird der Index der bearbeiteten Daten um die maximale Anzahl an Bytes

des WSTRINGS und die 4 Längen-Bytes erhöht. Sollten noch weitere empfangene Telegramme im Byte-Array vorhanden sein, werden diese auf dieselbe Weise umgewandelt. Ein Vorteil an C# ist die Unterstützung von Unicode.

Die Telegramme sind wie in Abbildung 18 gezeigt aufgebaut und als Trennzeichen wird „|“ verwendet. Die „MessageID“ wird zur Identifizierung des Telegramms verwendet.

```

1 //Get correct Eventtype
2 switch ((EventType)eventType)
3 {
4     case EventType.BatchEvent:
5         BatchLoggingEvent batchLoggingEvent = ConvertBatchDataToEvent(receivedData);
6         _sqlCommunication.WriteBatchLoggingData(batchLoggingEvent);
7         break;
8     case EventType.ConsumptionEvent:
9         ConsumptionLoggingEvent consumptionLoggingEvent =
10            ConvertConsumptionDataToEvent(receivedData);
11        _sqlCommunication.WriteConsumptionLoggingData(consumptionLoggingEvent);
12        break;
13    case EventType.DosageEvent:
14        DosageLoggingEvent dosageLoggingEvent = ConvertDosageDataToEvent(receivedData);
15        _sqlCommunication.WriteDosageLoggingData(dosageLoggingEvent);
16        break;
17    case EventType.HandlerEvent:
18        HandlerLoggingEvent handlerLoggingEvent = ConvertHandlerDataToEvent(receivedData);
19        _sqlCommunication.WriteHandlerLoggingData(handlerLoggingEvent);
20        break;
21    case EventType.ModuleEvent:
22        ModuleLoggingEvent moduleLoggingEvent = ConvertModuleDataToEvent(receivedData);
23        _sqlCommunication.WriteModuleLoggingData(moduleLoggingEvent);
24        break;
25    case EventType.RunEvent:
26        RunLoggingEvent runLoggingEvent = ConvertRunDataToEvent(receivedData);
27        _sqlCommunication.WriteRunEventData(runLoggingEvent);
28        break;
29    //NEW
30    case EventType.ProcessValueEvent:
31        List<ProcessValueLoggingEvent> processValueLoggingEvents =
32            ConvertProcessValueDataToEvents(receivedData);
33        _sqlCommunication.WriteProcessValueData(processValueLoggingEvents);
34        break;
35    case EventType.AlarmEventPlc:
36        AlarmPlcLoggingEvent alarmLoggingPlcEvent =
37            ConvertAlarmPlcDataToEvent(receivedData);
38        _sqlCommunication.WriteAlarmLoggingPlcData(alarmLoggingPlcEvent);
39        break;
40    //NEW EPM
41    case EventType.RunEventEpm:
42        RunEpmLoggingEvent runEpmLoggingEvent = ConvertRunEpmDataToEvent(receivedData);
43        _sqlCommunication.WriteRunEpmEventData(runEpmLoggingEvent);
44        break;
45    default:
46        _service_logger.Error($"ERROR ({eventType}): EventType is not supported!");
47        break;
48 }

```

Codelisting 6: Switch-Case zur Auswahl der Korrekten Konvertierungsfunktion (C#)

Über eine Switch-Case-Anweisung (Codelisting 6) wird die zu verwendende Konvertierungsfunktion ausgewählt und anschließend die Daten mit den „Write“-Funktionen der

SQLCommunication-Klasse in die Datenbank geschrieben.

```

1  private DosageLoggingEvent ConvertDosageDataToEvent(string[] receivedData)
2  {
3      DosageLoggingEvent result = null;
4      string errorMessage = "";
5      if (receivedData.Length == 11)
6      {
7          //extract the needed parameters
8          //var messageId = messageStringParts[0];           //not needed
9          //var globalMessageCounter = messageStringParts[1]; //not needed
10         if (!DateTime.TryParse(receivedData[2], out DateTime timestamp))
11             errorMessage += $" , timestamp {receivedData[2]}";
12         if (!bool.TryParse(receivedData[3], out bool active))
13             errorMessage += $" , active {receivedData[3]}";
14         if (!int.TryParse(receivedData[4], out int chemistry))
15             errorMessage += $" , chemistry {receivedData[4]}";
16         if (!bool.TryParse(receivedData[5], out bool dosageok))
17             errorMessage += $" , dosageok {receivedData[5]}";
18         if (!int.TryParse(receivedData[6], out int dosagetype))
19             errorMessage += $" , dosagetype {receivedData[6]}";
20         if (!int.TryParse(receivedData[7], out int dosagesubtype))
21             errorMessage += $" , dosagesubtype {receivedData[7]}";
22         if (!int.TryParse(receivedData[8], out int modulenumber))
23             errorMessage += $" , modulenumber {receivedData[8]}";
24         if (!double.TryParse(receivedData[9], NumberStyles.AllowDecimalPoint |
25             NumberStyles.AllowExponent, CultureInfo.InvariantCulture, out double
26             setpointvolume))
27             errorMessage += $" , setpointvolume {receivedData[9]}";
28         if (!double.TryParse(receivedData[10], NumberStyles.AllowDecimalPoint |
29             NumberStyles.AllowExponent, CultureInfo.InvariantCulture, out double volume))
30             errorMessage += $" , volume {receivedData[10]}";
31         //Check if Error while Parse
32         if (string.IsNullOrEmpty(errorMessage))
33             result = new DosageLoggingEvent(timestamp, active, chemistry, dosageok,
34                 dosagetype, dosagesubtype, modulenumber, setpointvolume, volume);
35     }
36     else
37     {
38         errorMessage += $"The Received-Data-Lenght is not Correct: {receivedData.Length},
39                         {receivedData}";
40     }
41     //Log Error-Message
42     if (!string.IsNullOrEmpty(errorMessage))
43         _service_logger.Error($"Error DosageData: {errorMessage}");
44     return result;
45 }
```

Codelisting 7: Konvertierung eines Dosage-Telegramm (C#)

RENA-Maschine müssen Flüssigkeiten dosieren, die Dosierung der Flüssigkeiten ist entscheidend für das Endergebnis der Produkte und wird deshalb mit dem Dosage-Telegramm mitgeloggt. Um dieses Telegramm zu konvertieren, wird die Funktion ConvertDosageData-

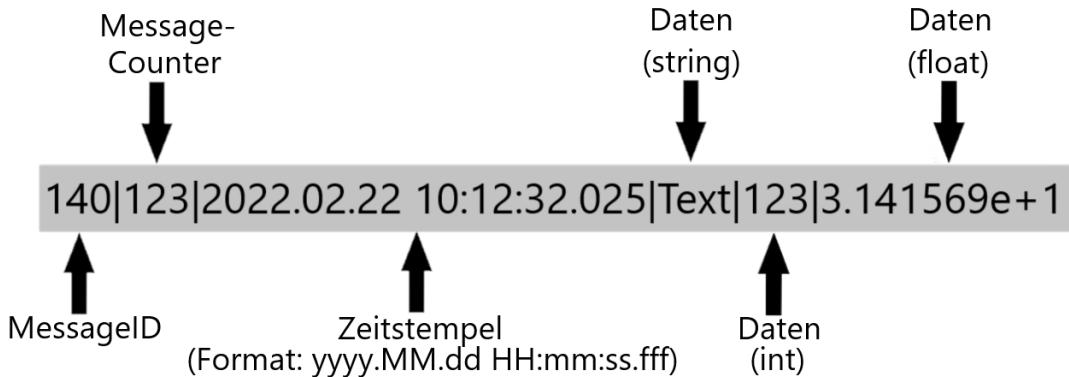


Abbildung 18: Telegramme-Aufbau

ToEvent-Funktion verwendet (Codelisting 7), bei den anderen Telegrammen ist der Funktion im Grunde dieselbe. Nach dem Anlegen des Rückgabewerts und des Fehler-Strings werden die eingehenden Daten auf ihre Länge überprüft (Zeile 5); sollten diese nicht der benötigten Länge entsprechen, wird der Fehler-String erweitert (Zeile 34). Die TryParse-Funktion überprüft, ob ein String in den gewünschten Datentyp umgewandelt werden kann und wandelt ihn wenn möglich um. Sollte der String nicht umgewandelt werden können, wird der Fehler-String erweitert. Sollten alle eingehenden Strings konvertiert worden sein ohne Fehlermeldung hervorzurufen, so wird ein neues Objekt erstellt (Zeile 29 und 30). In der Zeile 37 und 38 wird überprüft, ob der Fehlerstring nicht leer ist; sollte dies der Fall sein, wird dieser geloggt.

5.3 SQL-Kommunikation

Beim bestehenden Daten-Logging wird das PostgreSQL-Datenbanksystem [10] verwendet.

Mithilfe einer objektrelationaler Abbildung namens Dapper [13] können Objekte auf Datenbankrelationen abgebildet werden. Durch diese Abbildung können Daten einfach in die Datenbank eingefügt, gelesen oder verändert werden.

```

1 [Table(Table.NameConsumptionLogging)]
2
3 public class ConsumptionLoggingEventRow
4 {
5     [Key]
6     public long id { get; set; }
7     public DateTime datetime_local { get; set; }
8     public DateTime datetime { get; set; }
9     public int chemistryid { get; set; }
10    public double amount { get; set; }
11
12 }
```

Codelisting 8: Abbildungs-Objekt der Consumption-Tabelle (C#)

In Codelisting 8 wird für die Tabelle `NameConsumptionLogging` eine Klasse erstellt. In der Zeile 1 wird als „Table“-Argument der Tabellenname übergeben. Die Membervariablen müssen dieselben Namen haben wie die Spalten in der Tabelle. In der Zeile 5 wird der Variable „id“ das Argument Key übergeben, welches diese Variable als Primärschlüssel auszeichnet. Die Daten in der Spalte des Primärschlüssels müssen eindeutig sein, somit ist jede Zeile einzigartig und es gibt keine redundante Daten.

Die Tabellen werden entweder händisch erstellt oder man setzt das „SQL_AutoCreateTables“-Flag, so dass bei jedem Starten des Service überprüft wird, ob die benötigten Tabellen existieren und erstellt falls diese nicht existiert. Hierzu werden SQL-Scripte verwendet, für das Consumption-Logging beispielsweise das Script, welches in Codelisting 9 zu sehen ist.

```

1 CREATE TABLE IF NOT EXISTS public.{0}
2 (
3     id bigint NOT NULL DEFAULT nextval(''{0}_seq'::regclass),
4     datetime timestamp without time zone,
5     datetime_local timestamp without time zone NOT NULL,
6     chemistryid integer NOT NULL,
7     amount real NOT NULL,
8     CONSTRAINT {0}_pkey PRIMARY KEY(id)
9 )
10 TABLESPACE "PartitionD";
11 ALTER TABLE IF EXISTS public.{0}
12     OWNER to rena;
```

Codelisting 9: Create-Script für eine Consumption-Tabelle (SQL)

Im C#-Code wird dieses Script als String abgespeichert. Durch die Platzhalter „{0“ kann anschließend der korrekte Tabellenname eingesetzt werden. Hierzu wird in Zeile 3 für die

Spalte „id“ als Default-Wert die Funktion „nextval()“ aufgerufen mit der der nächste Wert ermittelt wird. Dafür muss in PostgreSQL eine Sequenz angelegt werden. Diese Sequence wird auch mit einem SQL-Script (Codelisting 10) erstellt; hier wird ebenfalls hier der Tabellenname in den Platzhalter eingefügt.

```

1 CREATE SEQUENCE IF NOT EXISTS public.{0}_seq
2   INCREMENT 1
3   START 1
4   MINVALUE 1
5   MAXVALUE 9223372036854775807
6   CACHE 1;
7
8 ALTER SEQUENCE public.{0}_seq
9   OWNER TO rena;
```

Codelisting 10: Create-Script für eine Sequence (SQL)

```

1 /// <summary>
2 /// Insert Consumption Data
3 /// </summary>
4 /// <param name="data">ConsumptionLoggingEvent</param>
5 public void WriteConsumptionLoggingData(ConsumptionLoggingEvent data)
6 {
7     if (data != null)
8     {
9         using (var conn = new NpgsqlConnection(connString))
10        {
11            try
12            {
13                //Connection open
14                conn.Open();
15                conn.Insert(data.GetLoggingRow());
16                if (_buffer.DataInBuffer)
17                    _buffer.WriteBufferData(conn);
18                conn.Close();
19            }
20            catch (NpgsqlException e)
21            {
22                _buffer.BufferingConsumptionLoggingEvent(data);
23                //Error logging
24                _sql_logger.Error($" Error WriteConsumptionLoggingData:
25                                {Helper.RemoveSpecialCharacters(e.Message)}");
26            }
27        }
28    }
```

Codelisting 11: Write-Funktion des Consumption-Logging (C#)

id [PK] bigint	datetime time without time zone	datetime_local time without time zone	chemistryid integer	amount real

Abbildung 19: Tabelle für das Consumption-Logging

Die Write-Funktionen (Codelisting 11), welche aus dem Switch-Case-Konstrukt aufgerufen werden, bauen eine Verbindung zur Datenbank auf und schreiben die Daten mithilfe von `Dapper` und `Npgsql` in die PostgreSQL-Datenbank. In Zeile 7 wird abgefragt, ob Daten existieren oder NULL sind. Sollten Daten existieren, wird eine neue Verbindung aufgebaut (Zeile 9). Nach dem Öffnen der Verbindung (Zeile 14) werden die Daten von einem Event-Objekt noch in ein EventRow-Objekt (Dapper-Klasse) umgewandelt und anschließend eingefügt (Zeile 15). Sollten noch Daten im SQL-Puffer sein, werden diese im Anschluss ebenfalls eingefügt. Danach wird die Verbindung wieder geschlossen. Sollte hierbei etwas schief gehen, werden die Daten im Puffer gespeichert und es wird eine Fehlermeldung geloggt.

Der SQL-Puffer ist so ausgelegt, eine bestimmte Anzahl an Daten zwischengespeichert werden, bevor diese verworfen werden. Sollte eine Verbindung hergestellt worden sein und noch Daten im Puffer sind, werden alle Daten aus dem Puffer eingefügt. Dies kann einen kurzen Ausfall der PostgreSQL-Datenbank überbrücken ohne einen Datenverlust zu erleiden.

6 RENA CSV Service

6.1 SQL-Kommunikation

Ebenfalls wie bei dem TCP-Logging werden auch bei dem CSV-Report die Klassen `Npgsql` und `Dapper` verwendet. Allerdings wird zusätzlich eine Trigger-Funktion auf der Datenbank erstellt (Codelisting 12).

```

1 DO $$ 
2 BEGIN
3 CREATE FUNCTION public."NotifyProcessEnd"()
4 RETURNS trigger
5 LANGUAGE 'plpgsql'
6 AS $BODY$ 
7 DECLARE
8 datarow RECORD;
9 BEGIN
10 IF(TG_OP = 'INSERT' AND TG_TABLE_NAME = 'modulelogging') THEN
11     datarow = NEW;
12     IF(datarow.moduleaction = 11) THEN --Process Finished
13         PERFORM pg_notify('productionend', 'finished');
14     ELSIF(datarow.moduleaction = 12) THEN --Process Aborted
15         PERFORM pg_notify('productionend', 'aborted');
16     ELSIF(datarow.moduleaction = 10) THEN --Process Started
17         PERFORM pg_notify('productionend', 'started');
18     END IF;
19 END IF;
20 RETURN NEW;
21 END
22 $BODY$;
23 ALTER FUNCTION public."NotifyProcessEnd"()
24 OWNER TO rena;
25 EXCEPTION
26 WHEN duplicate_function THEN --Ignore Duplicate-Function-Exception
27 END; $$
```

Codelisting 12: Create-Script für die Trigger-Funktion (SQL)

Diese Trigger-Funktion sendet eine Benachrichtigung auf dem „productionend“-Kanal mit der Information, wie der Prozess beendet wurde. Diese Funktion sendet nur bei den Modul-Aktionen *Prozess gestartet*, *Prozess beendet* oder *Prozess abgebrochen* eine Nachricht. Bei allen anderen Änderungen in der Datenbank wird keine Benachrichtigung gesendet.

```
1 DO $$  
2 BEGIN  
3     CREATE TRIGGER "OnInsertData"  
4         AFTER INSERT  
5         ON public.modulelogging  
6         FOR EACH ROW  
7             EXECUTE PROCEDURE public."NotifyProcessEnd"();  
8     EXCEPTION  
9         WHEN duplicate_object THEN --Ignore Duplicate-Function-Exception  
10    END; $$
```

Codelisting 13: Script für die Zuweisung der Trigger-Funktion zur Tabelle (SQL)

Diese Funktion wird über ein SQL-Skript (Codelisting 13) einer Tabelle zugewiesen. Hier wird die Tabelle „modulelogging“ (Zeile 5) verwendet.

6.2 Auswertung

Nachdem die Benachrichtigung empfangen, wird die Funktion `CreateCSVReport()` (Codelisting 14) aufgerufen. Diese sucht ein Start- und Enddatum in den Datensätzen. Sollte beides gefunden werden (keine Maximalwerte), so wird eine CSV-Datei erstellt und Daten geschrieben. Nachdem alle Daten geschrieben wurden, wird das Start-Datum des Reports gespeichert und in das Ini-File geschrieben, um doppelte Reports zu vermeiden. Daraufhin wird noch überprüft, ob ein weiterer Report erstellt werden kann.

```

1 public void CreateCSVReport()
2 {
3     //Check if Process is finished
4     DateTime[] ProcessDates = GetMinTimeSpanBetweenStartStop(_lastCSVReport);
5
6     while (ProcessDates[1] != DateTime.MaxValue)
7     {
8         _main_logger.Info($"Report Dates: Start-Date: {ProcessDates[0]}, End-DateTime:
9             {ProcessDates[1]}");
10        //Check if two Process Startes in Database, then is the first ProcessFinished
11        if (ProcessDates[0] != DateTime.MaxValue)
12        {
13            try
14            {
15                //Create header and get filename
16                string fileName = WriteHeaderGetFilename(ProcessDates[0]);
17                //Write all recipe-step-data from this report
18                WriteAllRecipeStepData(fileName, ProcessDates[0], ProcessDates[1]);
19                //Write EndStep
20                WriteEndReport(fileName, ProcessDates[1]);
21                _lastCSVReport = ProcessDates[0];
22                _iniFile.Write("LastCSVReportDate",
23                    HelperFunction.Helper.GetDateTimeStringWithMs(_lastCSVReport));
24                //Delete Files which are older than the _cleaningTimeSpan
25                _csvCleaner.DeleteOldFiles();
26            }
27            catch (Exception ex)
28            {
29                _main_logger.Info(ex.ToString());
28                break;
29            }
30        }
31        //Check if need to Create new Report
32        ProcessDates = GetMinTimeSpanBetweenStartStop(_lastCSVReport);
33    }
34 }
```

Codelisting 14: Funktion CreateCSVReport, welche die Reports erstellt (C#)

Die Prozessschritt-Zeiträume werden über die Prozessschritt-Start-Datensätze erfasst. Anschließend werden alle Daten, welche im Zeitraum vom Prozessschritt-Start bis zum nächsten Prozessschritt-Start erfasst wurden, in den CSV-Report geschrieben (Codelisting 15).

```

1  private void WriteAllRecipeStepData(string filename, DateTime from, DateTime to)
2  {
3      DateTime recipeStepStart = from;
4      while(recipeStepStart < to)
5      {
6          //Get actual and next RecipeStepStart
7          DateTime[] recipeStepStarts =
8              GetMinTimeSpanBetweenRecipeStepStart(recipeStepStart);
9          //If no RecipeStepStart exists, break dataWriter
10         if(recipeStepStarts[0] == DateTime.MaxValue || recipeStepStarts[0] > to)
11             break;
12         int stepNumber = GetRecipeStepNumber(recipeStepStarts[0]);
13         //Get and Write Module Data
14         var moduleData = _sqlCommunication.GetModuleLoggingEvents(recipeStepStarts[0],
15             recipeStepStarts[0]);
16         _csvWriter.WriteModuleData(filename, moduleData, _cultureInfo);
17         //Write all Recipe-Step-Data
18         if(recipeStepStarts[1] == DateTime.MaxValue || recipeStepStarts[1] > to)
19             WriteRecipeStepData(filename, stepNumber, recipeStepStarts[0], to);
20         else
21             WriteRecipeStepData(filename, stepNumber, recipeStepStarts[0],
22                 recipeStepStarts[1]);
23     }
}

```

Codelisting 15: Funktion zum schreiben aller Prozess-Schritt-Daten (C#)

6.3 CSV-Datei

Eine CSV-Datei ist eine Textdatei, welche zur Speicherung oder zum Austausch strukturierter Daten verwendet wird. CSV-Dateien werden meist für Tabellen verwendet, wobei ein Trennzeichen verwendet wird; im Fall des CSV-Reports ist es das Semikolon „;“. In der ersten Zeile kann der Spaltennamen gespeichert werden. CSV-Dateien können in verschiedene Programme (z.B. Excel) importiert werden. Mit der Filter-Funktion von Excel können die Daten dann besser ausgewertet werden. Selbstverständlich kann der Kunde die Daten auch in sein eigenes Auswertungssystem einlesen.

Für das Erstellen, Bearbeiten und Überprüfen einer Datei wird die Klasse `System.IO.File` verwendet.

Jede Dapper-Klasse, welche die EPM-Maschine verwendet, hat eine öffentliche Funktion mit dem Namen `GetLoggingEventCSVLine`. Diese gibt ein String zurück mit den Angaben für eine Zeile im CSV-Report.

```
1 public void WriteDosageData(string filename, IEnumerable<DosageLoggingEvent>
2     dosageLoggingEvents, string step, CultureInfo cultureInfo)
3 {
4     string CompleteFilename = GetCompleteFilename(filename);
5     var csvData = new StringBuilder();
6     foreach (DosageLoggingEvent dosageLoggingEvent in dosageLoggingEvents)
7     {
8         csvData.AppendLine(dosageLoggingEvent.GetLoggingEventCSVLine(step, cultureInfo));
9     }
10    File.AppendAllText(CompleteFilename, csvData.ToString());
}
```

Codelisting 16: Funktion zum schreiben der Dosage-Daten (C#)

Dieser String wird anschließend einem `StringBuilder` angehängt (Codelisting 16). Die Reports sollen in verschiedenen Sprachen erstellt werden können. Hierzu wird die Klasse `CultureInfo` verwendet, sie übernimmt die Formatierung der Datumsangaben und Zahlen für die ausgewählte Sprache. Weiterhin werden mittels einer *Multi-Language*-Datei (welches primär von RENA-WPFs verwendet wird) und einer Datenbank für die Maschinenkonfiguration Übersetzungen erzeugt. Mithilfe des Ini-Files wird die Sprache, in welcher der Report erstellt werden soll, festgelegt. Standard ist Englisch und aktuell wird nur Englisch (US) und Deutsch (DE) unterstützt, allerdings können durch einen kleinen Aufwand Sprachen hinzugefügt werden.

7 SPS-Logging

Das SPS-Logging greift auf Datenpunkte in der SPS zu; dies soll in einem konfigurierbaren Leseintervall durchgeführt werden. Daher wird ein Timer verwendet. Wenn die Zeit abgelaufen ist, wird wie im Programmablaufplan (Abbildung 20) gezeigt vorgegangen. Sollte das vorherige Lesen noch nicht beendet sein, so wird das aktuelle Lesen der Daten übersprungen um die Netzwerkbelastung nicht zu erhöhen.

7.1 SPS-Zugriff

7.1.1 AGLink Kommunikationsbibliothek

Für den Zugriff auf die SIEMENS-SPS wird Kommunikationsbibliothek AGLink von DELTA LOGIC verwendet. Diese wird als *Dynamic Link Library* (DLL) ausgeliefert.

Die Kommunikationsbibliothek wird in der Klasse `PLCCommunication` verwendet. Im Konstruktor (Codelisting 17) wird die Kommunikationsbibliothek durch einen Funktionsaufruf aktiviert (Zeile 4). Anschließend wird der Parameter-Pfad, in welchem die Kommunikation-Konfigurationsdateien liegen, in die Kommunikationsbibliothek geladen. Die Konfigurationsdateien benötigt die Bibliothek, um Verbindungen zur SPS aufzubauen. Zur Erstellung dieser Dateien gibt es von DELTA LOGIC ein extra Programm (Abbildung 21). Da bei RENA die SPS immer dieselbe IP-Adresse hat, können die erstellten Konfigurationsdateien für jede Maschine übernommen werden.

```

1 public PLCCommunication()
2 {
3     //Activate AGLink
4     AGL4.Activate(_activateKey);
5     _plc_logger.Info($"AGL activated!");
6
7     //Set ParaPath
8     string FullPath = Assembly.GetExecutingAssembly().Location;
9     string path = Path.GetDirectoryName(FullPath) + @"\Config\AGDeviceConfig";
10    AGL4.SetParaPath(path);
11
12    _plc_logger.Info($"AGL4 Parameter-Path: {path}");
13 }
```

Codelisting 17: Inititalisierung der PLC-Communication (C#)

Ein Zugriff auf ein Datenpunkt kann bei SIEMENS entweder über eine absolute Adresse oder über einen symbolischen Namen erfolgen. Bei optimierten Bausteinen kann nur noch über den symbolischen Namen zugegriffen werden, ein solchen Zugriff wird „symbolischer Zugriff“ genannt. Um einen symbolischen Zugriff durchführen zu können, muss zuerst das

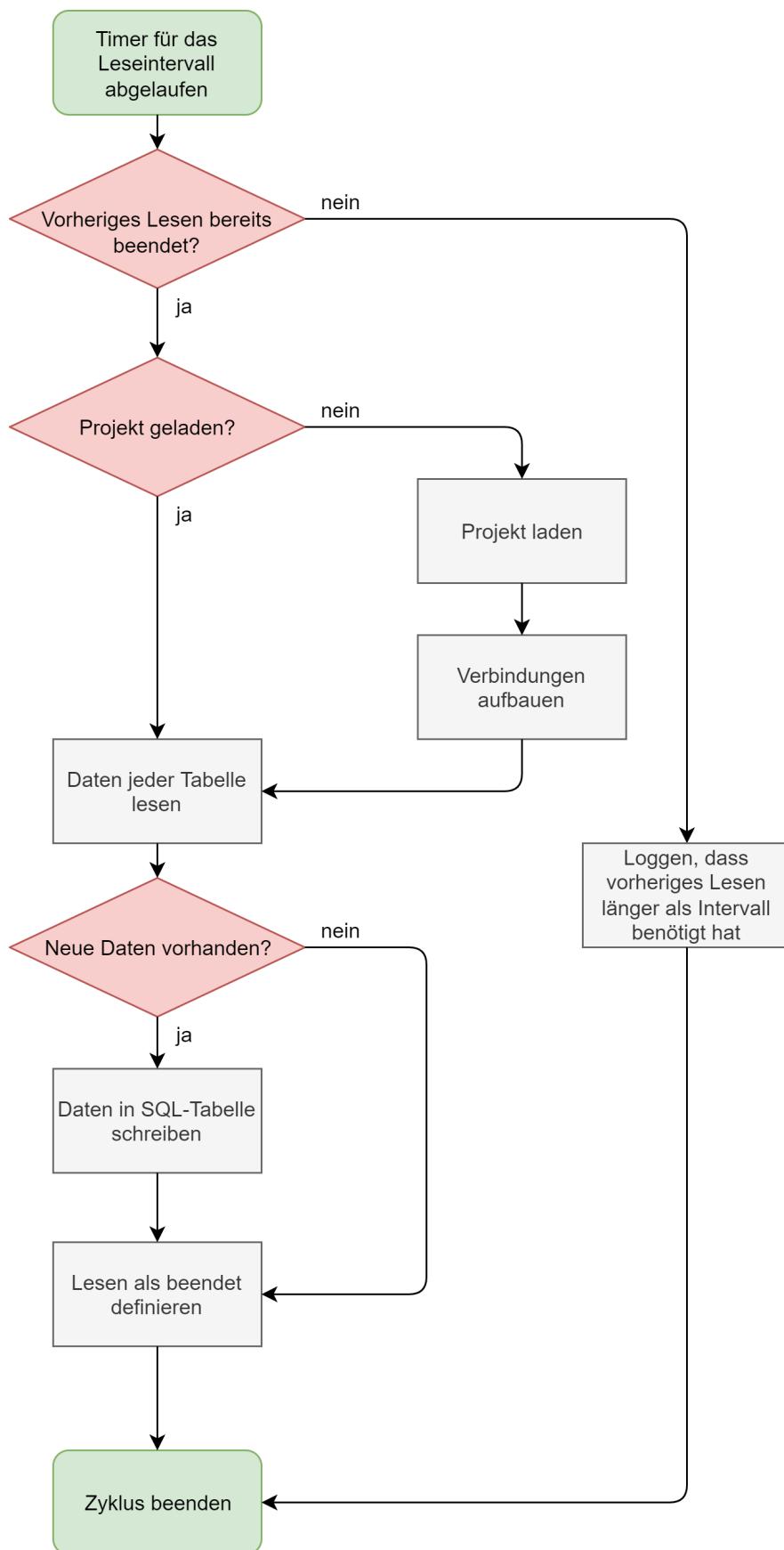


Abbildung 20: Programmablaufplan SPS-Logging

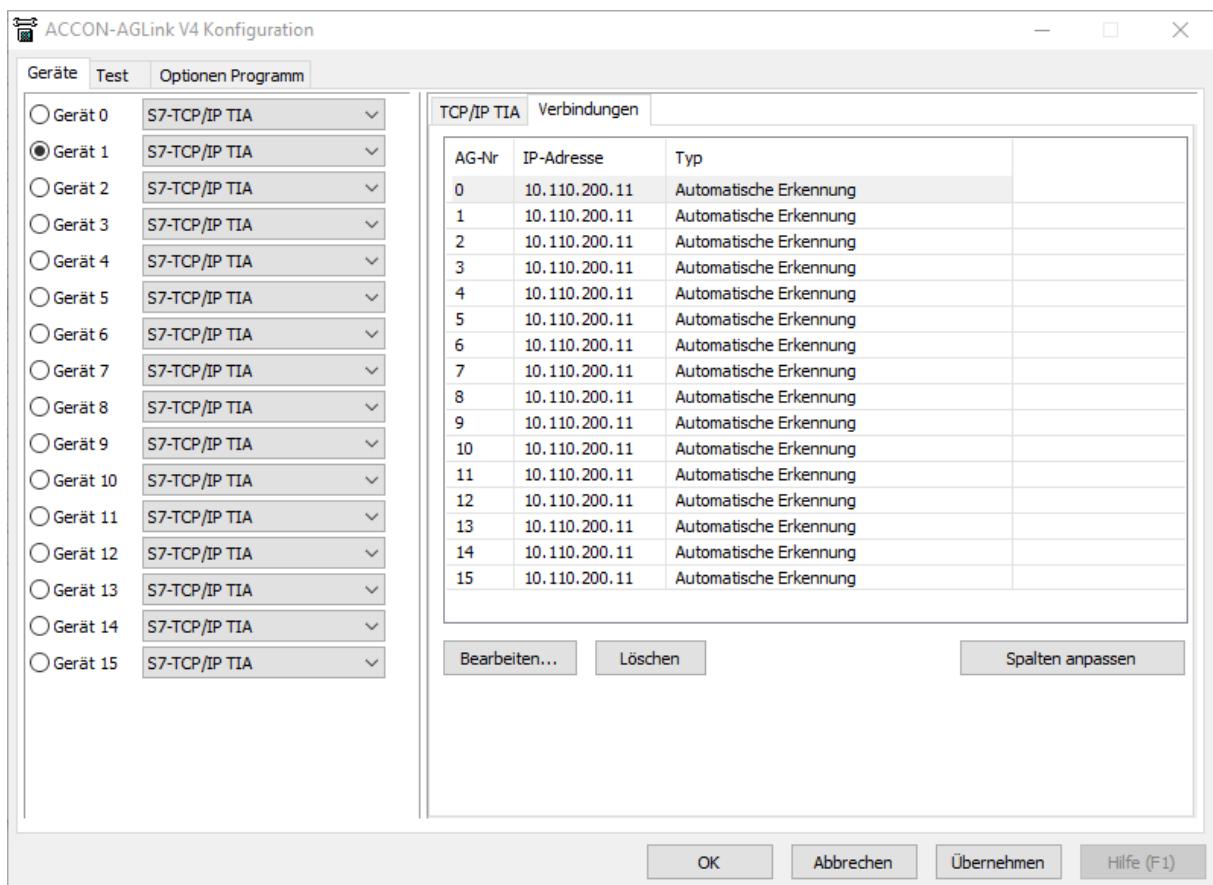


Abbildung 21: ACCON-AGLink Kommunikationskonfigurator

Projekt, welches auf der SPS gespeichert ist, geladen werden (Codelisting 18). Hierzu bietet die Bibliothek drei Möglichkeiten: Laden von einer TIA-Portal-Datei, Laden von einer AGL-Datei oder online laden von der SPS. Es wurde entschieden, dass das Projekt online geladen werden soll. Dadurch ist das geladene Projekt immer aktuell. Somit muss auch keine TIA-Portal-Datei oder AGL-Datei auf der Maschine abgelegt werden, wodurch das Know-How der RENA geschützt wird. Die **AGParas**, welche als Parameter übergeben werden, sind die Parameter (Zeile 5), welche die Bibliothek für die Kommunikation nutzt. Sollte das Projekt nicht geladen werden können, so wird eine Fehlermeldung in die Log-Datei des Service geschrieben (Zeile 13). Diese Log-Datei kann zur Erkennung von Fehlerfällen innerhalb des Service genutzt werden.

```

1  ///<summary>
2  /// Open Project
3  ///</summary>
4  ///<param name="paras"></param>
5  public bool openProject(ref AGParas paras)
6  {
7      int result = 0;
8
9      result = AGL4.Symbolic_LoadAGLinkSymbolsFromPLC(paras.Agl.ConnNr, ref
10     paras.RootHandle);
11
12     if (result != AGL4.AGL40_SUCCESS)
13     {
14         _plc_logger.Error($"Could not open Project with ConnectionNumber
15             {paras.Agl.ConnNr}! ErrorCode: {result.ToString("X")}; ErrorText:
16             {paras.Agl.GetErrorMsg(result)}");
17         return false;
18     }
19     return true;
20 }
```

Codelisting 18: Öffnen eines Projekts von der SPS (C#)

DELTA LOGIC empfiehlt, mehrere Verbindungen zu einer SPS aufzubauen, sollte man viele Daten lesen wollen. Daher wird pro Tabelle eine Verbindung aufgebaut. Für jede Verbindung werden universelle Parameter benötigt. Mit der Funktion **GetNewAgParas** (Codelisting 19) werden den einzelnen Verbindungen universelle Parameter zugewiesen. Hierzu werden

```

1  /// <summary>
2  /// Get new AGParameters
3  /// </summary>
4  /// <returns>AGParas</returns>
5  public AGParas GetNewAGParas()
6  {
7      AGParas agParas = new AGParas();
8      //Create TIA Instance
9      agParas.Agl = AGL4ConnectionFactory.CreateTiaInstance(_globalDeviceCounter,
10         _globalAGCounter, 200);
11     //Logging enable?
12     agParas.Verbose = false;
13     agParas.ConnectionErrorLogged = false;
14     _globalAGCounter++;
15     if(_globalAGCounter > 15)
16     {
17         _globalDeviceCounter++;
18         _globalAGCounter = 0;
19         if(_globalDeviceCounter > 15)
20         {
21             plc_logger.Error("GetNewAGParas(): too many AGParas needed!! Could not create
22               new Connection!!!!");
23             _globalAGCounter = 16;
24             _globalDeviceCounter= 16;
25         }
26     }
27
28     return agParas;
29 }
```

Codelisting 19: Zuweisen der Verbindungsparameter (C#)

Nach der Zuweisung der Verbindungsparameter werden die Verbindungen geöffnet (Codelisting 20). Hierzu wird in Zeile 7 überprüft, ob eine Verbindung bereits aufgebaut wurde. Sollte dies der Fall sein, so muss nichts weiter gemacht werden. Ansonsten wird mit der Funktion `paras.Agl.Connect()` in Zeile 9 die Verbindung aufgebaut. Als Rückgabewert liefert die Funktion einen Wahrheitswert. Sollte eine Verbindung aufgebaut worden sein, so wird dies in das Log-File des Services geschrieben; des Weiteren wird die `ConnectionErrorLogged`-Flag zurückgesetzt. Dieses Flag wird benutzt, damit eine Fehlverbindung nur einmal in das Log-File geschrieben wird (Zeile 19).

```

1  /// <summary>
2  /// Check if Agl is connected, if not connect to device
3  /// </summary>
4  /// <param name="paras"></param>
5  public void connectDevice(ref AGParas paras)
6  {
7      if (!paras.Agl.Connected)
8      {
9          if (paras.Agl.Connect())
10         {
11             _plc_logger.Info($"Connected to {paras.Agl.DevNr} Device and {paras.Agl.PlcNr}
12             PLC.");
13             if (paras.ConnectionErrorLogged)
14                 paras.ConnectionErrorLogged = false;
15         }
16         else
17         {
18             if (!paras.ConnectionErrorLogged)
19             {
20                 _plc_logger.Error($"Could not connect to {paras.Agl.DevNr} Device and
21                 {paras.Agl.PlcNr} PLC! ErrorCode:
22                 {paras.Agl.ConnErrorCode.ToString("X")}] ErrorText:
23                 {paras.Agl.ConnErrorMessage}");
24                 paras.ConnectionErrorLogged = true;
25             }
26         }
27     }
28 }

```

Codelisting 20: Verbindung aufbauen (C#)

Für jede Variable, welche gelesen wird, wird ein Struct erstellt (Codelisting 21). Diese Structs werden in einem Buffer gespeichert. Dieser wird mit der Funktion `createBuffer()` (Codelisting 22) reserviert. Für den Zugriff auf ein Symbol wird ein Int-Pointer benötigt; dieser wird in der Zeile 13 zugewiesen. Anschließend wird die Buffergröße bestimmt (Zeile 24). Sollte hierbei ein Fehler auftreten, so wird das in das SError-Feld des Symbols gespeichert (Zeile 36). Somit kann beim Auswerten des Symbols direkt eine Fehlermeldung ausgegeben werden. Anschließend wird in Zeile 38 ein neues Bytearray mit der benötigten Größe angelegt. Des Weiteren wird die Buffergröße ebenfalls im Symbol abgespeichert (Zeile 39). Dies wird für alle Symbole durchgeführt.

```
1 public struct SymbolicRW
2 {
3     public IntPtr AccessHandle;
4
5     public byte[] Buffer;
6
7     public int BufferLen;
8
9     public int Result;
10
11    public int SError;
12 }
```

Codelisting 21: Struct für das Lesen einer Variable (C#)

```

1  /// <summary>
2  /// Create Buffer for Reading Data
3  /// </summary>
4  private int createBuffer(ref AGL4.SymbolicRW[] symbolicRWs, AGParas agParas, string[]
5   symbols)
6  {
7      int result = AGL4.AGL40_SUCCESS;
8      for (int i = 0; i < symbols.Length; i++)
9      {
10         bool ErrorFlag = false;
11         // First create an access object using the path to the desired node.
12         // With these objects we can access the plcs data.
13         int errorPosition = 0;
14         result = AGL4.Symbolic_CreateAccessByPath(agParas.RootHandle, symbols[i], ref
15             symbolicRWs[i].AccessHandle, ref errorPosition);
16         if (result != AGL4.AGL40_SUCCESS)
17         {
18             if (agParas.Verbose)
19             {
20                 _plc_logger.Error($"CreateBuffer Error CreateAccessByPath for
21                     \\"{symbols[i]}\\": {result} -> {agParas.Agl.GetErrorMsg(result)}");
22             }
23             ErrorFlag = true;
24         }
25         // Now with the access object we can get the memory amount that we have to
26         // allocate for the data.
27         int bufferSize = 0;
28         result = AGL4.Symbolic_GetAccessBufferSize(symbolicRWs[i].AccessHandle, ref
29             bufferSize);
30         if (result != AGL4.AGL40_SUCCESS)
31         {
32             if (agParas.Verbose)
33             {
34                 _plc_logger.Error($"CreateBuffer Error GetAccessBufferSize for
35                     \\"{symbols[i]}\\": {result} -> {agParas.Agl.GetErrorMsg(result)}");
36             }
37             ErrorFlag = true;
38         }
39         if (ErrorFlag)
40         {
41             symbolicRWs[i].SError = ErrorNumbers.SYMBOLRW_BUFFER_NOT_CREATED;
42         }
43         symbolicRWs[i].Buffer = new byte[bufferSize];
44         symbolicRWs[i].BufferLen = bufferSize;
45     }
46     return result;
47 }
```

Codelisting 22: Buffer erstellen (C#)

Das Lesen der Symbole wird mit der Funktion `textttSymbolic_ReadMixEx()` (Codelisting 23 Zeile 2) ausgeführt. Sollte dabei ein Fehler auftreten, so wird dieser in einem

Popup-Fenster angezeigt und in das Log-File geschrieben. Sollte kein Fehler auftreten, so werden die gelesenen Daten von der Funktion `texttEvaluateData()` (Zeile 20) ausgewertet. Auch hier wird im Fehlerfall ein Popup-Fenster angezeigt. Diese Funktion wertet die gelesenen Daten nacheinander aus und speichert sie in den einzelnen Logging-Items. Bei der Datenauswertung werden der `ValueType` und der `SystemType` zur Bestimmung des Datentyps ausgewertet. Dadurch kann das Byte-Array aufbereitet werden. Hierzu stellt AGLink für jeden SIEMENS Datentyp Funktionen zur Verfügung. Anschließend wird noch überprüft, ob das Symbol auf Wertänderung geloggt werden soll; sollte dies der Fall sein, so wird das Flag `writeTableToSQL` gesetzt. Ist dieses Flag gesetzt, so wird die Logging-Tabelle in der SQL-Datenbank abgespeichert.

```

1 //Read Data
2 result = loggingTable.agParas.Agl.Symbolic_ReadMixEx(symbolicRWs);
3 if (result != AGL4.AGL40_SUCCESS)
4 {
5
6     //Error
7     loggingTable.tableLoggingError = true;
8     if (!loggingTable.tableLoggingErrorShow)
9     {
10         string caption = $"Could not Read Data";
11         string message = $"Could not read data.\n Errorcode:
12             0x{result.ToString("X")}\nError message:
13             {loggingTable.agParas.Agl.GetErrorMsg(result)}";
14         _plc_logger.Error($"SHOW ERROR: CAPTION {caption}, MESSAGE {message}");
15         ErrorHandling.ErrorShow.showError(caption, message);
16         loggingTable.tableLoggingErrorShow = true;
17     }
18 }
19 //Evaluation
20 else
21 {
22     result = evaluateData(ref loggingTable, symbolicRWs);
23     if(result != AGL4.AGL40_SUCCESS)
24     {
25         //Error
26         loggingTable.tableLoggingError = true;
27         if (!loggingTable.tableLoggingErrorShow)
28         {
29             string caption = $"Could not Read Data";
30             string message = $"Could not evaluate Data.\nErrorcode:
31                 0x{result.ToString("X")}\nError message:
32                 {loggingTable.agParas.Agl.GetErrorMsg(result)}";
33             _plc_logger.Error($"SHOW ERROR: CAPTION {caption}, MESSAGE {message}");
34             ErrorHandling.ErrorShow.showError(caption, message);
35             loggingTable.tableLoggingErrorShow = true;
36         }
37     }
38 else
39 {
40     //Success reading Data, no Error raised
41     loggingTable.tableLoggingError = false;
42     loggingTable.tableLoggingErrorShow = false;
43 }
44 }
```

Codelisting 23: Lesen der Symbole (C#)

7.1.2 SIEMENS-Variablen

Die RENA-Maschinen sind in Module unterteilt, dadurch werden die Maschinen besser konfigurierbar und können ideal dem Kundenwunsch angepasst werden. Es gibt verschiedene Kategorien in welche die Module eingeteilt werden können: die Material-Ein- und

	Name	Datentyp	Startwert	Remanenz	Erreichbar a...	Schreib...	Sichtbar i...	Einstellwert	...
1	stMod	"ST_MOD_QDR_Base_HWMn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	stIB_BathOverfull	"ST_DigIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	bEnF	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
4	bFActive	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
5	bFOn	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
6	bRes4	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
7	bRes5	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
8	bRes6	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
9	bRes7	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
10	bRes8	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
11	bOn	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
12	bOn	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
13	bRes11	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
14	bRes12	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
15	bRes13	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
16	bRes14	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
17	bRes15	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
18	bRes16	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
19	stIW_LevelProcessTank	"ST_AnalogIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
20	wRaw	Word	16#0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
21	rNormed	Real	0.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
22	rFNormed	Real	0.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
23	bfActive	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
24	bEnF	Bool	false		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
25	stIW_ConductivityProcessTank	"ST_AnalogIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
26	stIW_ConductivitySensorTemp	"ST_AnalogIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
27	stCompPropValveControl	"ST_SdCompPropFlow_HWMn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
28	stSlaveOpenFeedback	"ST_AnalogIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
29	stActFlow	"ST_AnalogIn"				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
30						<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Abbildung 22: Datenbaustein mit Beispielvariablen

Ausgabestationen, das Handling und die Prozess-Becken. Das *Quick-Dump-Rinse* (QDR)-Becken ist ein Modul der Kategorie Prozess-Becken, welches zur Reinigung der Wafer¹ benutzt wird. Auch die Software für die SPS ist nach diesen Modulen strukturiert.

Variablen werden bei SIEMENS in Datenbankbausteinen gespeichert. Als Beispiel wird hier der Datenbankbaustein „GDB_MOD_QDR_1_HWIN“ (Abbildung 22) aufgeführt. Dieser Datenbaustein wird für die Eingänge des QDR-Moduls verwendet, da es mehrere QDR-Module in einer RENA-Maschine geben kann, wird eine Struktur für die Eingänge und Ausgänge eines QDR-Moduls angelegt. AGLink benötigt für das Auslesen nun einen sogenannten Tagname. Dieser Tagname wird in der Maschinenkonfigurationsdatenbank abgelegt. Er setzt sich zusammen aus dem Präfix „PLC.Blocks.“, gefolgt vom Datenbausteinnamen sowie den nachfolgenden Struktornamen bis hin zum Variablennamen. Als Beispiel:

PLC.Blocks.GDB_MOD_QDR_1_HWIN.stMod.stIB_BathOverfull.bEnF ist der vollständige Tagname des ersten Bool-Werts im Beispieldatenbaustein. Für die Umwandlung der SIEMENS-Datentypen in .NET-Framework Datentypen und schließlich in PostgreSQL-Datentypen wird Maschinenkonfiguration (Abbildung 23) verwendet.

¹quadratische oder runde, dünne Silizium-Scheiben in der Mikroelektronik, Photovoltaik und Mikrosystemtechnik

TagName	LoggingInterval	TableName	VarID	LoggingActive	DataType	Value type	Description	Module type	ModuleTypeIndex	Component	Unit	RideDesignation	Unit	DefaultColor	LanguageID
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stW_LevelProcessTank.rNormed	10 logging_qdr	var1	WAHR real	AnalogIn	Level Sensor	QDR	1 Module	1 Module	0/1	0/1	%	IW.1006	E60690	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWOut.stMod.stQY_DrainTo_IC_OHW.bQOn	0 logging_qdr	var10	WAHR boolean	DigitalOut	Valve Drain	QDR	1 Module	1 Module	0/1	0/1	0/1	Q340.4	89AC76	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWOut.stMod.stCompPropValveControl.stY_OpenSupply.bQOn	0 logging_qdr	var11	WAHR boolean	DigitalOut	Supply Valve DIW-20	QDR	1 Module	1 Module	0/1	0/1	0/1	Q340.3	64SF31	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stB1_BathOverfull.bQOn	0 logging_qdr	var12	WAHR boolean	DigitalIn	Overflow Sensor	QDR	1 Module	1 Module	0/1	0/1	0/1	I120.4	642424	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stW_ConductivityProcess.Tank_Normed	10 logging_qdr	var2	WAHR real	AnalogIn	Conductivity Sensor	QDR	1 Module	1 Module	0/1	0/1	0/1	IW.024	CDA34	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stControlstFlow_Normed	10 logging_qdr	var3	WAHR real	AnalogIn	Flow Sensor	QDR	1 Prop. Valve Control	1 Prop. Valve Control	0/1	0/1	0/1	I/min	IW.008	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stCompPropValveControl.stFlow_Normed	10 logging_qdr	var4	WAHR real	AnalogIn	Feedback Prop. Valve	QDR	1 Prop. Valve Control	1 Prop. Valve Control	0/1	0/1	0/1	%	IW.010	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stMot.stMot.stTemp.Normed	10 logging_qdr	var5	WAHR real	AnalogIn	Conductivity Sensor Act. Temperature	QDR	1 Module	1 Module	0/1	0/1	0/1	-C	CC0605	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stCompPropValveControl.QW_SetpointPropValve.tQOut	0 logging_qdr	var6	WAHR real	AnalogOut	Setpoint Prop. Valve	QDR	1 Prop. Valve Control	1 Prop. Valve Control	0/1	0/1	0/1	%	QW1014	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWin.stMod.stB1_BathOverfull.bQOn	0 logging_qdr	var7	WAHR boolean	DigitalIn	Overflow Sensor	QDR	1 Module	1 Module	0/1	0/1	0/1	I120.4	642424	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWOut.stMod.stQY_SprayBar.bQOn	0 logging_qdr	var8	WAHR boolean	DigitalOut	Valve Spray Bar	QDR	1 Module	1 Module	0/1	0/1	0/1	0/1	4E5754	1033	
PLC.Blocks.GDB.MOD.QDR.1.HWOut.stMod.stQY_QuickDump.bQOn	0 logging_qdr	var9	WAHR boolean	DigitalOut	Valve Quick Dump	QDR	1 Module	1 Module	0/1	0/1	0/1	IW.005	605532	1033	

Abbildung 23: Maschinenkonfiguration für ein QDR-Modul

7.2 SQL-Kommunikation

Wie beim TCP-Logging wird auch hier dieselbe Datenbank PostgreSQL-Datenbank verwendet. Allerdings kommt Dapper nicht zum Einsatz, da die Tabellen je nach Maschinenkonfiguration verschieden aufgebaut sind. Aus der Maschinenkonfiguration (Abbildung 23) werden die Spalten „VarID“, „TableName“ sowie „DataType“ für das Erstellen der Tabellen verwendet. Um den SQL-Befehl zu erzeugen wird die Funktion `GetCreateTableString()` (Codelisting 24) verwendet. Diese Funktion erhält als Übergabeparameter die `LoggingTable`, aus welcher der SQL-Befehl zusammengesetzt wird. Die Funktion `GetColumnCreateString` (Codelisting 25), welche in Zeile 10 aufgerufen wird, ist für die Teilbefehl der Variablen verantwortlich. Hierbei wird für jedes `LoggingItem` die „VarID“ als Spaltenname verwendet (Zeile 12). Zusätzlich wird zuvor der PostgreSQL-Datentyp ermittelt (Zeile 9). Des Weiteren dürfen alle Spalten, welche für Variablen vorgesehen sind, NULL sein. Sollte zukünftig eine Variable aus der Maschinenkonfiguration für das Logging entfallen, so kann diese einfach weiter mit NULL befüllt werden.

```

1  ///<summary>
2  /// Generate Create String from loggingTable
3  ///</summary>
4  public static string GetCreateTableString(LoggingTable loggingTable)
5  {
6      string begin = $"CREATE TABLE public.{loggingTable.TableName} \n(\n";
7      string primaryKey = $"id bigint NOT NULL DEFAULT
8          nextval('{loggingTable.TableName}_seq)::regclass),";
9      string datetimes = "datetimestamp timestamp without time zone,\n datetime_local timestamp
10         without time zone NOT NULL,\n";
11      //Add Variables
12      string vars = GetColumnCreateString(loggingTable.Items);
13
14      string end = $"      CONSTRAINT {loggingTable.TableName}_pkey PRIMARY KEY (id) USING
15          INDEX TABLESPACE \"PartitionD\" ) TABLESPACE \"PartitionD\"; ALTER TABLE
16          public.{loggingTable.TableName} OWNER to rena;";
17
18      return begin + primaryKey + datetimes + vars + end;
19 }
```

Codelisting 24: Erstellen des CreateTable-Befehl (C#)

```

1  /// <summary>
2  /// Generate Column String for all LoggingItems
3  /// </summary>
4  private static string GetColumnCreateString(List<LoggingItem> items)
5  {
6      StringBuilder result = new StringBuilder();
7      foreach (LoggingItem item in items)
8      {
9          var sqlDataType = GetSQLDatatype(item.DataType);
10         if(sqlDataType != null)
11         {
12             string column = $"{item.VarID} {sqlDataType},\n";
13             result.Append(column);
14         }
15         else
16         {
17             _sql_logger.Error($"Could not generate CreateString for Column! VarID:
18                 {item.VarID} Tag: {item.LoggingTag} DataType: {item.DataType}");
19         }
20     }
21     return result.ToString();
}

```

Codelisting 25: Erstellen der CreateColumn-Befehls (C#)

Das im Lasten-/Pflichtenheft aufgeführte Überprüfen der Tabelle wird von zwei Funktion übernommen. Die Überprüfung ob Tabelle überhaupt existiert führt die Funktion `CheckIfTableExist()` (Codelisting 26) aus. Als erstes wird die Ergebnisvariable „result“ initialisiert und auf den Errorwert -1 gesetzt. Anschließend wird eine neue `NpgsqlConnection` erstellt (Zeile 9), welche in Zeile 13 geöffnet wird. Anschließend wird der SQL-Befehl (Codelisting 27) ausgeführt. Mit dem SELECT-Statement in Zeile 3 wird abgefragt, ob eine Tabelle existiert. Das SELECT EXISTS in Zeile 2 wandelt das Ergebnis in einen Integer-Wert um, da in Zeile 8 dieser Datentyp angegeben ist. Sollte eine Tabelle existieren, so wird in Zeile 15 eine 1 in die Ergebnisvariable gespeichert, ansonsten eine 0. Umschlossen wird das Öffnen und Ausführen des Befehls mit einem Try-Catch-Block (Zeile 11 und 17), um Fehler abzufangen. Sollte ein Fehler auftreten, so wird dieser in das Logging-File geschrieben und die Ergebnisvariable wird auf -1 gesetzt. Zum Schluss wird die Verbindung geschlossen (Zeile 25) und die Ergebnisvariable zurückgegeben (Zeile 27).

```

1  /// <summary>
2  /// Check if Table Exist
3  /// </summary>
4  /// <param name="tablename">Tablename</param>
5  /// <returns> 0 = not exist, 1 = exist, -1 = Error</returns>
6  public int CheckIfTableExist(string tableName)
7  {
8      int result = -1;
9      using(var conn = new NpgsqlConnection(connString))
10     {
11         try
12         {
13             conn.Open();
14             //run query
15             result =
16                 conn.QuerySingle<int>(String.Format(SQLScripts.CheckIfTableExist,tableName));
17         }
18         catch(NpgsqlException e)
19         {
20             //Error logging
21             _sql_logger.Error($" Error Check If TableExist: " +
22                 Helper.RemoveSpecialCharacters(e.Message));
23             //Return Error
24             result = -1;
25         }
26         //Connection close
27         conn.Close();
28     }
29     return result;
30 }
```

Codelisting 26: Überprüfung der SQL-Tabelle (C#)

```

1  -- Check if {0}-Table exists in Database
2  SELECT EXISTS (
3      SELECT FROM
4          pg_tables
5      WHERE
6          schemaname = 'public' AND
7          tablename  = '{0}'
8      );
```

Codelisting 27: Überprüfung der SQL-Tabelle (SQL)

Mit einer ähnlichen Funktion werden die Spalten überprüft, allerdings wird hierbei zusätzlich der Datentyp überprüft. In Zeile 2 bis 4 in Codelisting 28 wird getestet, ob in der Tabelle die Spalte existiert und in Zeile 8 bis 10 wird der Datentyp der Spalte zurückge-

geben. Dieser wird anschließend mit dem geforderten Datentyp abgeglichen.

```
1 -- Check if {1}-Column exists in {0}-Table
2 SELECT EXISTS (SELECT column_name
3 FROM information_schema.columns
4 WHERE table_name = '{0}' AND column_name = '{1}');
5
6
7 -- Get Datatype of {0}-Column from {1}-Table
8 SELECT pg_typeof({0})::text
9 FROM public.{1}
10 LIMIT 1;
```

Codelisting 28: Überprüfung der SQL-Spalten (SQL)

Um Daten in die Datenbank zu schreiben, wird die Funktion `WriteData()` (Codelisting 29) verwendet. Hierfür werden der SQL-INSERT-Befehl zusammengesetzt und die Daten als DynamicParameters (Zeile 12) übergeben. Zunächst wird der Anfang des INSERT-Befehls gespeichert (Zeile 14), dabei wird der Tabellenname übergeben. Anschließend werden zwei String-Builder initialisiert (Zeile 16,17). String-Builder werden für performante String-Manipulationen eingesetzt. Diese String-Builder können beispielsweise mit `.Append` weitere Strings anhängen, ohne dabei neuen Speicherplatz zu reservieren. In Zeile 19 bis 22 werden der aktuelle Zeitstempel und der *Coordinated Universal Time* (koordinierte Weltzeit) (UTC)-Zeitstempel den Parametern und dem INSERT-Befehl angehängt. Danach wird jedes Logging-Item den Parametern und INSERT-Befehl hinzugefügt (Zeile 24 bis 29). In der Zeile 31 und 32 werden die Endklammern der Parameter und der Werte hinzugefügt, sowie das Semikolon für das Ende des INSERT-Befehls. Danach wird der gesamte Befehls-String zusammengesetzt (Zeile 34). Nach Öffnen der SQL-Verbindung (Zeile 37) wird mit dem Execute-Befehl der Insert-Befehl mit den Werten als Parameter ausgeführt (Zeile 39). Durch den Catch-Block (Zeile 41) werden Fehler abgefangen und direkt in die Logging-Datei des Service geschrieben (Zeile 44). Zuletzt wird die SQL-Verbindung geschlossen (Zeile 47). Dies wird nach dem Catch-Block durchgeführt, damit auch in einem Fehlerfall die Verbindung sicher geschlossen wird.

```

1  /// <summary>
2  /// Write Data to LoggingTable
3  /// </summary>
4  /// <param name="loggingTable"> Data</param>
5  public void WriteData(LoggingTable loggingTable)
6  {
7      using (var conn = new NpgsqlConnection(connString))
8      {
9          try
10         {
11             //Generate Dynamic Parameters
12             var para = new DynamicParameters();
13             //Create INSERT-Command Begin
14             string begin = $"INSERT INTO public.{loggingTable.TableName}(";
15             //Create StringBuilder for Parameters and Values
16             StringBuilder paras = new StringBuilder();
17             StringBuilder values = new StringBuilder();
18             //Add DateTime and DatetimeLocal
19             paras.Append("datetime, datetime_local");
20             values.Append($"VALUES ( @datetime, @datetime_local ");
21             para.Add("datetime_local", DateTime.Now);
22             para.Add("datetime", DateTime.UtcNow);
23             //Adding every LoggingItem and Parameter
24             foreach (LoggingItem item in loggingTable.Items)
25             {
26                 paras.Append($",{item.VarID}");
27                 values.Append($",@{item.VarID}");
28                 para.Add(item.VarID, item.Value);
29             }
30             //Append Ending
31             paras.Append(")");
32             values.Append(");");
33             //Concat complete INSERT Command
34             string insert_cmd = begin + paras.ToString() + values.ToString();
35
36             //Open Connection
37             conn.Open();
38             //Execute Command
39             conn.Execute(insert_cmd, para);
40         }
41         catch (Exception e)
42         {
43             //Error logging
44             _sql_logger.Error($" Error Insert Data: " +
45                         Helper.RemoveSpecialCharacters(e.Message));
46         }
47         //Close Connection
48         conn.Close();
49     }
}

```

Codelisting 29: Daten in Datenbank schreiben (C#)

7.3 HMI-Kommunikation

Die HMI-Kommunikation wird benutzt, um Informationen zwischen dem Service und der Visualisierung der Maschine auszutauschen. Aktuell benutzt wird nur die Kommunikationsrichtung *Visualisierung zu Service*, allerdings wird die Richtung *Service zu Visualisierung* ebenfalls eingebunden. Als Kommunikationstechnologie wird TCP eingesetzt, benutzt werden die Ports 1948 und 1949.

7.3.1 *Visualisierung zu Service*

Dieser Kommunikationskanal mit dem Port 1948 wird benutzt, um den Service zu steuern. Es wird der gleiche TCP-Server verwendet wie bei dem TCP-Logging. Über Event-Types kann ein Telegramm identifiziert werden. Unterstützt wird in der aktuellen Version nur der Event `ReloadData`, dabei werden die MachineConfig neu eingelesen, alle SQL-Tabellen überprüft und gegebenenfalls verändert oder neu erstellt, sowie die Verbindungen zur SPS neu aufgebaut. Sollte der Event-Type nicht identifiziert werden können, so werden die empfangenen Daten in das Logging-File geschrieben (Codelisting 30 in Zeile 28, 41 und 52). Die Switch-Case-Anweisung in Zeile 20 bis 31 kann einfach erweitert werden, sollten noch weitere Steuerungsmöglichkeiten für den Service hinzukommen.

```

1  /// <summary>
2  /// Evaluation of the received data from HMI
3  /// </summary>
4  /// <param name="sender"></param>
5  /// <param name="receivedData">Received Data in a string array</param>
6  private void _hmiCommunication_ReceivedDataHandler(object sender, string[] receivedData)
7  {
8      StringBuilder mgs = new StringBuilder();
9      for(int i = 0; i < receivedData.Length; i++)
10     {
11         mgs.Append($" [{i}]: {receivedData[i]};");
12     }
13     //Check if HMIEventTypes is a int
14     if (int.TryParse(receivedData[0], out var eventType))
15     {
16         //Check if HMIEventTypes is known
17         if (Enum.IsDefined(typeof(HMIEventTypes), eventType))
18         {
19             //Get correct HMIEventTypes
20             switch ((HMIEventTypes)eventType)
21             {
22                 case HMIEventTypes.ReloadData:
23                     _main_logger.Info("Reload MachineConfig, received over HMICommunication");
24                     ReloadData();
25                     break;
26                 default:
27                 {
28                     _main_logger.Info("HMICommunication received Data, but cant evaluate:
29                         " + mgs.ToString());
30                 }
31                 break;
32             }
33             //Error is Eventtype not defined
34         }
35     }
36     _main_logger.Error("HMICommunication received Data, HMIEventTypes is not
37         defined: " + mgs.ToString());
38 }
39 //Error is Eventtype not a int
40 else
41 {
42     _main_logger.Error("HMICommunication received Data, HMIEventTypes is not a int: "
43         + mgs.ToString());
44 }

```

Codelisting 30: Empfangene Daten von der Visualisierung auswerten (C#)

7.3.2 Service zu Visualisierung

Aktuell wird diese Kommunikationsrichtung noch nicht verwendet, allerdings soll sie in Zukunft eingesetzt werden, um Fehlermeldungen des Services auf der Visualisierung anzuzeigen. Die Funktion `SendData()` (Codelisting 31) sendet den übergebenen String an die übergebene IP-Adresse über den Port 1949. Dafür wird ein neuer Socket erzeugt (Zeile 10), welcher in Zeile 13 mit der übergebenen IP-Adresse und dem Port verbunden wird. Anschließend wird der String unter Benutzung der Encoding-Klasse in ein Byte-Array umgewandelt (Zeile 15), hierbei wird die UTF-8-Kodierung verwendet. Anschließend wird das Byte-Array über den Socket gesendet (Zeile 17) und der Socket wieder geschlossen (Zeile 19). Durch den Try-Catch-Block (Zeile 8 und 22) werden Fehler erkannt und diese werden dann in das Log-File des Service geschrieben (Zeile 24).

```

1  /// <summary>
2  /// Send String-Message to Destination-IP
3  /// </summary>
4  /// <param name="ipAddress">destination IP-Adress</param>
5  /// <param name="message">Message</param>
6  public void SendData(IPAddress ipAddress, string message)
7  {
8      try
9      {
10         using(Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
11             ProtocolType.Tcp))
12         {
13             //Connect Socket to ipAdress with sending port
14             socket.Connect(ipAddress, _sendingPort);
15             //Convert Message to UTF8-Bytes
16             byte[] bytes = Encoding.UTF8.GetBytes(message);
17             //Send Data
18             socket.Send(bytes);
19             //Close Socket
20             socket.Close();
21         }
22     }
23     catch (Exception e)
24     {
25         _hmi_logger.Error($"Could not send Data to HMI. IP: {ipAddress}
26             Port:{_sendingPort} Message: {message} Exception: {e}");
27     }
28 }
```

Codelisting 31: Daten per TCP an Visualisierung senden (C#)

8 Ausblick

Die Logging-Services, TCP-Logging und SPS-Logging sollen in Zukunft auf vielen Maschinen eingesetzt werden. Durch die einfache Installation können diese auch auf bereits ausgelieferte Maschinen, je nach Kundenwunsch, nachgerüstet werden.

8.1 TCP-Logging

Das TCP-Logging ist dank der Telegramme flexibel gestaltet und kann durch eine Telegrammerweiterung immer den Maschinen angepasst werden. Durch diese Flexibilität kann der Service bei Sondermaschinen sowie Standardmaschinen verwendet werden.

Da eine TCP-Kommunikation eingesetzt wird können, auch andere SPSen von weiteren Herstellern mit dem Service kommunizieren und Daten mitloggen. Dadurch könnten alle RENA-Maschinen mit diesem Service ausgestattet werden.

8.2 SPS-Logging

Aktuell werden nur SIEMENS-SPSen unterstützt. Eine Ausweitung des Service auf SPSen anderer Hersteller wäre durchaus denkbar. Somit könnten mehr RENA-Maschinen mit dem SPS-Logging-Service ausgestattet werden.

Des Weiteren wurde das Tabellenformat in der Datenbank vom bisherigen Tool „Accon EasyLog“ übernommen, da die Auswertungsfunktionalität der Daten bereits existiert. Allerdings sind die Tabellen nicht platzsparend aufgebaut. Wie in der Beispieltabelle (Abbildung 24) zu sehen ist, werden immer alle Datenpunkte pro Zeile gespeichert, auch wenn sich nur ein einzelner Wert ändert. Optimieren könnte man den Tabellenaufbau, indem man die Tabelle aufteilt in solche Datenpunkte, welche auf Wertänderung geloggt werden und solche, die in einem bestimmten Intervall geloggt werden. Dadurch werden redundante Daten vermieden und Speicherplatz gespart.

	id	[PK] bigint	datetime	timestamp without time zone	datetime_local	timestamp without time zone	var1	var2	var3	var4	var5	var6	var7	var8	var9	var10	var11	var12
1	1163	2022-08-09 08:30:04.273611		2022-08-09 08:30:04.273611		1.025	24.13	0	0	31.41	3.141	true	false	true	true	true	true	
2	1162	2022-08-09 08:30:01.259375		2022-08-09 08:30:01.259375		1.025	24.13	0	0	31.41	3.141	true	false	false	false	false	false	
3	1161	2022-08-09 08:29:59.241823		2022-08-09 08:29:59.241823		1.025	24.13	0	0	31.41	3.141	true	false	true	true	true	true	
4	1160	2022-08-09 08:29:58.231305		2022-08-09 08:29:58.231305		1.025	24.13	0	0	31.41	3.141	true	false	false	false	false	false	
5	1159	2022-08-09 08:29:48.164908		2022-08-09 08:29:48.164908		1.025	24.13	0	0	31.41	3.141	true	false	false	false	false	false	
6	1158	2022-08-09 08:29:38.088998		2022-08-09 08:29:38.088998		1.025	24.12	0	0	31.41	3.141	true	false	false	false	false	false	
7	1157	2022-08-09 08:29:28.028642		2022-08-09 08:29:28.028642		1.025	0	0	0	31.41	3.141	true	false	false	false	false	false	
8	1156	2022-08-09 08:29:17.978424		2022-08-09 08:29:17.978424		1.025	0	0	0	31.41	3.141	true	false	false	false	false	false	
9	1155	2022-08-09 08:29:07.938667		2022-08-09 08:29:07.938667		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
10	1154	2022-08-09 08:28:57.896719		2022-08-09 08:28:57.896719		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
11	1153	2022-08-09 08:28:51.824964		2022-08-09 08:28:51.824964		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
12	1152	2022-08-09 08:28:49.810514		2022-08-09 08:28:49.810514		42.42	0	0	0	31.41	3.141	true	false	false	true	true	true	
13	1151	2022-08-09 08:28:47.792865		2022-08-09 08:28:47.792865		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
14	1150	2022-08-09 08:28:45.760288		2022-08-09 08:28:45.760288		42.42	0	0	0	31.41	3.141	true	false	true	true	true	true	
15	1149	2022-08-09 08:28:37.71639		2022-08-09 08:28:37.71639		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
16	1148	2022-08-09 08:28:27.649596		2022-08-09 08:28:27.649596		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
17	1147	2022-08-09 08:28:17.596342		2022-08-09 08:28:17.596342		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	
18	1146	2022-08-09 08:28:16.500451		2022-08-09 08:28:16.500451		42.42	0	0	0	31.41	3.141	true	false	false	false	false	false	

Abbildung 24: Beispiel Tabelle

9 Literatur

Literatur

- [1] *DELTA LOGIC*. <https://www.deltalogic.de/>, . – [Online; Zugriff Juli 2022]
- [2] *Automatisierungspyramide*. <https://www.wachendorff-prozesstechnik.de/technologie/feldbus/>, . – [Online; Zugriff September 2022]
- [3] *Microsoft .NET*. <https://dotnet.microsoft.com/en-us/>, . – [Online; Zugriff Februar 2022]
- [4] BEER, Wolfgang ; BRINGRUBER, Dietrich ; MÖSSENBÖCK, Hanspeter ; WÖSS, Albrecht: *Die .NET-Technologie*. 1. dpunkt.verlag, 2003. – ISBN 3-89864-174-0
- [5] *WPF - Windows Presentation Foundation*. <https://docs.microsoft.com/de-de/visualstudio/designers/getting-started-with-wpf?view=vs-2022>, . – [Online; Zugriff Januar 2022]
- [6] *Microsoft-Website: C#*. <https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/>, . – [Online; Zugriff Januar 2022]
- [7] SCHÄPERS, Arne ; HUTTARY, Rudolf ; BREMES, Dieter: *C# Kompendium*. Markt+Technik Verlag, 2004. – ISBN 3-8272-6773-0
- [8] *TCP-Kommunikation Elektronik-Kompendium*. <https://www.elektronik-kompendium.de/sites/net/2009211.htm>, . – [Online; Zugriff März 2022]
- [9] *TCP-Standard RFC793*. <https://datatracker.ietf.org/doc/html/rfc793>, . – [Online; Zugriff März 2022]
- [10] *PostgreSQL-Website*. <https://www.postgresql.org/>, . – [Online; Zugriff Januar 2022]
- [11] *ISO 7498-1 Website*. <https://www.iso.org/standard/20269.html>, . – [Online; Zugriff März 2022]
- [12] *Quelle des OSI-Modell-Bild*. <https://www.cloudflare.com/de-de/learning/ddos/glossary/open-systems-interconnection-model-osi/>, . – [Online; Zugriff März 2022]
- [13] *Github von Dapper*. <https://github.com/DapperLib/Dapper>, . – [Online; Zugriff Januar 2022]

10 Anhang

10.1 RENA CSV Report als CSV-Datei

D:\EXPORT\RenaCSVReport\2022_02_18_09_47_12_Run ID Test Logger_ProductionReport.csv

Montag, 7. März 2022 09:12

```

Date;TimeStamp;Event;Step;Parameter;Value;Unit
2/18/2022;9:47:12 AM;LotID;;Run ID Test Logger;;CRLF
2/18/2022;9:47:12 AM;Recipe;;recipename;;CRLF
2/18/2022;9:47:12 AM;Process started;0;;CRLF
2/18/2022;9:47:13 AM;Process step started;0;;CRLF
2/18/2022;9:47:13 AM;Process parameter;0;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:13 AM;Process parameter;0;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:13 AM;Process parameter;0;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:13 AM;Process parameter;0;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:13 AM;Process parameter;0;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:13 AM;Process parameter;0;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:14 AM;Alarm;0;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:13 AM;Dosage;0;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:13 AM;Dosage;0;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:14 AM;Process step started;1;;CRLF
2/18/2022;9:47:14 AM;Process parameter;1;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:14 AM;Process parameter;1;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:14 AM;Process parameter;1;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:14 AM;Process parameter;1;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:14 AM;Process parameter;1;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:14 AM;Process parameter;1;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:15 AM;Alarm;1;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:15 AM;Dosage;1;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:15 AM;Dosage;1;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:15 AM;Process step started;2;;CRLF
2/18/2022;9:47:16 AM;Process parameter;2;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:16 AM;Process parameter;2;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:16 AM;Process parameter;2;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:16 AM;Process parameter;2;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:16 AM;Process parameter;2;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:16 AM;Process parameter;2;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:16 AM;Alarm;2;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:16 AM;Dosage;2;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:16 AM;Dosage;2;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:17 AM;Process step started;3;;CRLF
2/18/2022;9:47:17 AM;Process parameter;3;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:17 AM;Process parameter;3;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:17 AM;Process parameter;3;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:17 AM;Process parameter;3;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:17 AM;Process parameter;3;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:17 AM;Process parameter;3;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:17 AM;Alarm;3;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:17 AM;Dosage;3;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:18 AM;Dosage;3;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:18 AM;Process step started;4;;CRLF
2/18/2022;9:47:18 AM;Process parameter;4;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:18 AM;Process parameter;4;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:18 AM;Process parameter;4;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:19 AM;Process parameter;4;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:19 AM;Process parameter;4;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:19 AM;Process parameter;4;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:19 AM;Alarm;4;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:19 AM;Dosage;4;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:19 AM;Dosage;4;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:20 AM;Process step started;5;;CRLF
2/18/2022;9:47:20 AM;Process parameter;5;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:20 AM;Process parameter;5;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:20 AM;Process parameter;5;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:20 AM;Process parameter;5;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:20 AM;Process parameter;5;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:20 AM;Process parameter;5;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:21 AM;Alarm;5;GEN Control Off;Warning;Came inCRLF
2/18/2022;9:47:20 AM;Dosage;5;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:20 AM;Dosage;5;Automatic DI_COLD By level;3,1414999961853;sec.CRLF
2/18/2022;9:47:21 AM;Process step started;6;;CRLF
2/18/2022;9:47:21 AM;Process parameter;6;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:21 AM;Process parameter;6;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:21 AM;Process parameter;6;Flow circulation end;123,458000183105;l/minCRLF
2/18/2022;9:47:21 AM;Process parameter;6;Steptime;123,458000183105;sec.CRLF
2/18/2022;9:47:21 AM;Process parameter;6;Flow circulation start;123,458000183105;l/minCRLF
2/18/2022;9:47:21 AM;Process parameter;6;Flow circulation end;123,458000183105;l/minCRLF

```

10.2 RENA CSV Report als Excel-Datei

Date	TimeStamp	Event	Step	Parameter	Value	Unit
2/18/2022	9:47:12 AM	LotID		Run ID Test Logger		
2/18/2022	9:47:12 AM	Recipe		recipename		
2/18/2022	9:47:12 AM	Process starte	0			
2/18/2022	9:47:13 AM	Process step s	0			
2/18/2022	9:47:13 AM	Process param	0	Steptime	123,4580002	sec.
2/18/2022	9:47:13 AM	Process param	0	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:13 AM	Process param	0	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:13 AM	Process param	0	Steptime	123,4580002	sec.
2/18/2022	9:47:13 AM	Process param	0	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:13 AM	Process param	0	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:13 AM	Dosage	0	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:13 AM	Dosage	0	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:14 AM	Alarm	0	GEN Control Off	Warning	Came in
2/18/2022	9:47:14 AM	Process step s	1			
2/18/2022	9:47:14 AM	Process param	1	Steptime	123,4580002	sec.
2/18/2022	9:47:14 AM	Process param	1	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:14 AM	Process param	1	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:14 AM	Process param	1	Steptime	123,4580002	sec.
2/18/2022	9:47:14 AM	Process param	1	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:14 AM	Process param	1	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:15 AM	Alarm	1	GEN Control Off	Warning	Came in
2/18/2022	9:47:15 AM	Dosage	1	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:15 AM	Dosage	1	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:15 AM	Process step s	2			
2/18/2022	9:47:16 AM	Process param	2	Steptime	123,4580002	sec.
2/18/2022	9:47:16 AM	Process param	2	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:16 AM	Process param	2	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:16 AM	Process param	2	Steptime	123,4580002	sec.
2/18/2022	9:47:16 AM	Process param	2	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:16 AM	Process param	2	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:16 AM	Alarm	2	GEN Control Off	Warning	Came in
2/18/2022	9:47:16 AM	Dosage	2	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:16 AM	Dosage	2	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:17 AM	Process step s	3			
2/18/2022	9:47:17 AM	Process param	3	Steptime	123,4580002	sec.
2/18/2022	9:47:17 AM	Process param	3	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:17 AM	Process param	3	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:17 AM	Process param	3	Steptime	123,4580002	sec.
2/18/2022	9:47:17 AM	Process param	3	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:17 AM	Process param	3	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:17 AM	Dosage	3	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:18 AM	Alarm	3	GEN Control Off	Warning	Came in
2/18/2022	9:47:18 AM	Dosage	3	Automatic DI_COLD By level	3,141499996	sec.
2/18/2022	9:47:18 AM	Process step s	4			
2/18/2022	9:47:18 AM	Process param	4	Steptime	123,4580002	sec.
2/18/2022	9:47:18 AM	Process param	4	Flow circulation start	123,4580002	l/min
2/18/2022	9:47:18 AM	Process param	4	Flow circulation end	123,4580002	l/min
2/18/2022	9:47:19 AM	Process param	4	Steptime	123,4580002	sec.
2/18/2022	9:47:19 AM	Process param	4	Flow circulation start	123,4580002	l/min